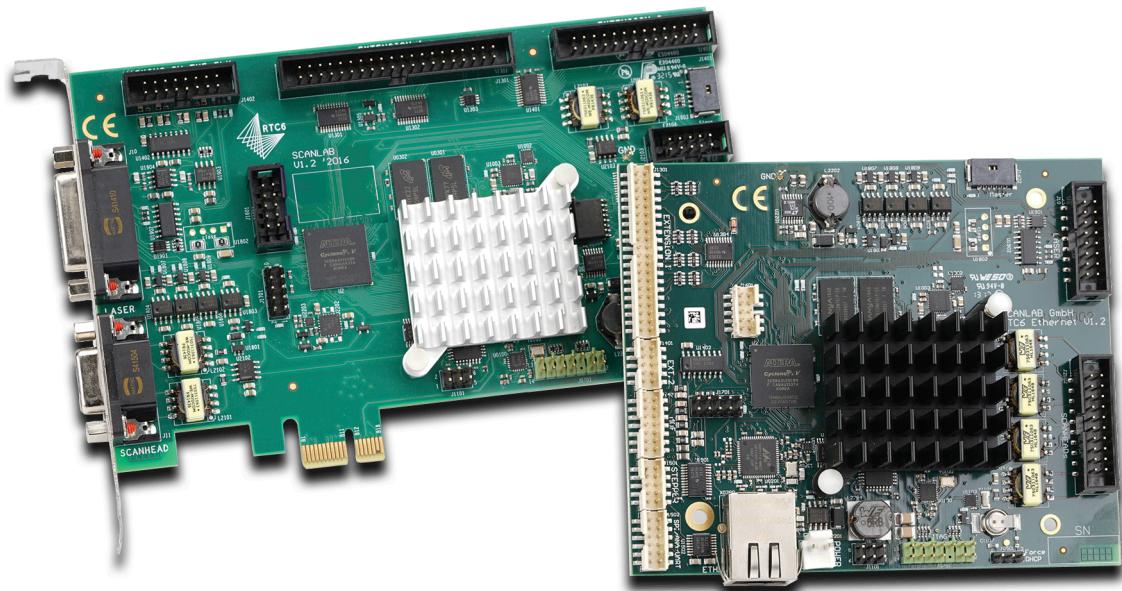




Installation and Operation

RTC6 PCIe Board RTC6 Ethernet Board

Real Time Control of Scan Systems and Lasers
RTC6 Software Package V1.14.1



SCANLAB GmbH
Siemensstr. 2a
82178 Puchheim
Germany

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

info@scanlab.de
www.scanlab.de

© SCANLAB GmbH
(Doc. Rev. 1.0.20 en-US - 2022-07-22)

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

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

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



Contents

1	About this Manual	24
1.1	Manufacturer	24
1.2	Related Documents	25
1.3	Glossary and Abbreviations	26
2	Product Overview	30
2.1	Labeling	30
2.2	Unpacking Instructions and Typical Scope of Delivery	30
2.3	Delivered RTC6 Software Package	30
	Folder Correction Files	31
	Folder CorrectionFileConverter	31
	Folder Demo Files	31
	Folder HPGL	31
	Folder iSCANCfg	31
	Folder RTC6 Driver	31
	Folder RTC6 Files	32
	Folder RTC6 Tools	33
2.4	Intended Use	34
2.5	System Requirements	36
2.5.1	Hardware	36
2.5.2	Software	36
2.6	Options	37
2.7	Jumper Settings and Type Designations	39
2.7.1	Solder Jumper Field A – Configuring the Output Signal Level at EXTENSION 1 Socket Connector	40
2.7.2	Solder Jumper Field B – Configuring Pin (17) of EXTENSION 2 Socket Connector	41
2.7.3	Solder Jumper Field C – Configuring Pin (15) of EXTENSION 2 Socket Connector	42
2.8	Accessories for the RTC6 PCIe Board	43
2.8.1	XY2-100 Converter	43
2.8.2	Laser Adapter	43
2.8.3	Data Cables	43
2.8.4	Slot Cover with 9-pin D-SUB Connector for "2. SCANHEAD" Socket Connector	44
2.8.5	Slot Cover with 15-pin D-SUB Connector for "MARKING ON THE FLY" Socket Connector	44
2.8.6	Slot Cover with 15-pin D-SUB Connector and 9-pin D-SUB Connector	45
2.8.7	Extension Board "RTC5/6 varioSCAN FLEX Extension"	45
2.9	Supplementary Software	46
2.10	Notes for RTC4 Users	47
2.10.1	Hardware Changes	47
	Controlling Scan Systems	47
	Controlling the Laser	47
	EXTENSION 1 Socket Connector	47
	EXTENSION 2 Socket Connector	48
	MARKING ON THE FLY Socket Connector	48
	Other Hardware Interfaces	48
2.10.2	Porting RTC4 Source Code to the RTC6 PCIe Board	48
	Changed Initialization	48
	Command Changes	49
	Increased Parameter Resolution	50
	Changed Timing Behavior	51
2.10.3	New and Changed Functionality	52



Interface to the PC	52
Controlling Scan Systems	52
Controlling the Laser	53
Interfaces for Peripheral Equipment	53
General Programming	54
Laser Marking	55
Special Functions	55
2.11 Notes for RTC5 Users	57
2.11.1 RTC6 PCIe Boards vs. RTC5 Boards	57
Functionality changes	57
Parameter changes	58
2.11.2 RTC6 PCIe Boards and Scan Systems with SCANahead Servo Control	58
2.11.3 Restrictions with RTC6 PCIe Boards	59
Availability of Technical Variants	59
Peripheral Interfaces	59
Functionality	59
2.12 Source Code for RTC6 User Programs	59
2.12.1 Creating New RTC6 Source Code	59
2.12.2 Adapting RTC5 Source Code for the RTC6 PCIe Board	59
Mandatory Steps	59
Optional Steps and Notes on Further Modification and Optimization	60
2.13 Changes to the RTC6 Command Set	61
2.13.1 Changes to the RTC6 Command Set	61
2.13.2 Removed from RTC6 Command Set	61
3 Safety During Installation and Operation	62
3.1 Safe Operation	62
3.2 Laser Safety	62
4 RTC6 PCIe Board – Layout and Interfaces	63
4.1 Layout – Upper Side	63
4.2 Layout – Lower Side	64
4.3 Interface to the PC	65
4.4 Master Socket Connector, Slave Socket Connector	65
4.5 Interfaces to Scan System	66
4.5.1 Scan Head Connectors and Transfer Protocol	66
SCANHEAD Connector	67
2. SCANHEAD Socket Connector	67
4.5.2 XY2-100 Converter (Accessory)	68
4.5.3 Data Cables (Accessories)	70
4.6 Interfaces for the Laser and Peripheral Equipment	72
4.6.1 LASER Connector	72
Laser Control Signals	72
External Control Signals	73
BUSY List Execution Status	73
2-Bit Digital Input Port	73
2-Bit Digital Output Port	73
12-Bit Analog Output Port 1 and 2	73
Input and Output Wiring	73
Laser Adapter (Accessory)	75
4.6.2 EXTENSION 1 Socket Connector	77
Configuring the Output Signal Level	77
16-Bit Digital Input Port and 16-Bit Digital Output Port	77



Synchronization of Data Acquisition	78
BUSY List Execution Status	78
4.6.3 EXTENSION 2 Socket Connector	79
Configuration by Solder Jumpers	79
Laser Control Signals	80
8-Bit Digital Output Port	80
4.6.4 MARKING ON THE FLY Socket Connector	81
Encoder Input Ports	81
External Control Signals	81
Analog Output Port	81
BUSY List Execution Status	81
MARKING ON THE FLY Slot Cover (Accessory)	82
4.6.5 RS232 Socket Connector	82
4.6.6 McBSP/ANALOG Socket Connector	83
McBSP Interface	83
Analog Input Ports	85
4.6.7 STEPPER MOTOR Socket Connector	86
5 Installation and Start-Up	87
5.1 Checking Jumper Settings	87
5.2 Installing the RTC6 PCIe Board	87
5.3 Installing the RTC6 Board Driver	88
5.4 Installing the RTC6 Software	89
5.5 Safe Start-up and Shutdown Sequences	90
5.6 Functionality Test	90
5.7 User Programs and Demo Programs	91
6 Developing RTC6-User Programs	92
6.1 RTC6 Software Concept Basics	92
6.1.1 Controlling Scan Systems and Lasers – An Introductory Example	92
6.1.2 Control Commands and List Commands	93
6.2 Initialization and Program Start-Up	94
6.2.1 DLL Calling Convention	94
6.2.2 Importing Commands	94
Pascal	95
C	95
C++	95
C#	95
6.2.3 Initializing the RTC6 DLL and RTC6 Board Management	96
6.2.4 Start of RTC6 PCIe Board Operation	97
Initializing the Board	97
Configuring the Board	97
Initializing the Scan System Control	98
Initializing the Laser Control	98
Loading and Executing Lists	98
6.2.5 Example Code (C)	99
6.3 RTC6 List Memory	101
6.3.1 Lists and the Protected List Memory Area	101
"List 1" and "List 2"	101
"List 3" – Protected List Memory Area	101
6.3.2 Configuring the RTC6 List Memory	102
6.4 List Handling	104
6.4.1 Loading Lists	104



"Unconditional" Loading	104
Loading with Protection	105
Terminating Lists	105
6.4.2 List Status	106
6.4.3 List Execution Status	107
6.4.4 Starting and Stopping Lists	108
6.4.5 Interrupting Lists for Synchronization of Processing	109
6.4.6 Changing Lists Automatically	109
One-Time List Change	109
Alternating List Changes	110
6.5 Structured Programming	111
6.5.1 Subroutines	111
Non-Indexed Subroutines	111
Indexed Subroutines	112
General Information on Calling Subroutines	113
Index Management and Defragmentation	114
Subsequent Protection and Conversion of Non-Indexed Subroutines	115
Unprotecting Subroutines	116
6.5.2 Character Sets and Text Strings	117
Defining Indexed Character Sets	117
Calling Indexed Characters	118
Defining Indexed Text Strings for Time, Date and Serial Number	118
Calling Indexed Text Strings	119
Managing Indexed Characters and Text Strings	119
6.5.3 Jumps	119
6.5.4 RTC4-Circular Queue Mode	120
6.5.5 Loops	121
6.6 Using Several RTC6 PCIe Boards in One PC	122
6.6.1 Multi-Board Programming	122
Examples (Pascal)	122
6.6.2 Single-Board Programming	123
6.6.3 Master/Slave Operation	123
Initialization with RTC6 Software Package \geq V1.5.0 (\geq RBF 619)	124
Initialization with RTC6 Software Package $<$ V1.5.0 (\leq RBF 618)	124
Synchronous Starts and Synchronous Stops	125
Example Code (Delphi)	126
6.7 Usage of RTC6 PCI Express Boards by Several User Programs	127
6.7.1 Notes on Board Acquisition by a User Program	127
6.8 Error Handling	129
6.8.1 Download Verification	130
6.8.2 Checking for Overruns	131
6.8.3 Example Code (C)	132
6.9 Miscellaneous	134
6.9.1 Free Variables	134
7 Basic Functions for Scan Head Control and Laser Control	135
7.1 Marking Dots, Lines and Arcs	135
7.1.1 Marking with Vector Commands and "Arc" Commands	135
Jump Commands	136
Mark Commands	136
Arc Commands	136
Polylines	136
Ellipse Commands	137



[*]Para[*] Commands	138
7.1.2 Microstepping	139
7.1.3 Marking Single Dots	140
7.1.4 Example Code (C)	141
7.2 Delay Settings – Coordinating Scan Head Control and Laser Control	144
7.2.1 Laser Delays	144
LaserOn Delay	145
LaserOff Delay	145
7.2.2 Scanner Delays	146
Jump Delay	146
Variable Jump Delays	147
Mark Delay	148
Polygon Delay	149
Variable Polygon Delays	150
User-defined “Variable Polygon Delays”	152
7.2.3 Notes on Optimizing the Delays	154
Recommended Sequence	154
Automatic Delay Adjustments	154
Potential Errors when Optimizing the Delays	156
7.2.4 Sky Writing	159
Sky Writing Mode 1	159
Sky Writing Mode 2	160
Sky Writing Mode 3	161
Synchronization	161
7.3 Scan Head Control	166
7.3.1 Reference System	166
7.3.2 Image Field Size and Image Field Calibration	166
Typical Image Field	167
Compatibility Modes	167
7.3.3 Virtual Image Field	168
Coordinate Transformations in the Virtual Image Field	168
7.3.4 3D Image Field	169
Carrying Out Adjustment	169
Checking the z axis Calibration	169
Testing 3-Axis Scan Systems with F-Theta Objective	171
7.3.5 Image field Correction and Correction Tables	172
Field Distortion	172
Image Field Correction Algorithm	173
Activating Image Field Correction	173
2D and 3D Correction Files	173
Loading Correction Tables	174
Assigning Loaded Correction Tables	174
1to1 Correction Tables	176
Inverse Tables	176
ct5 Correction File Header	177
Converting Correction Files	179
7.3.6 Output Values to the Scan System	180
Calculation	180
Value Ranges	180
Precalculation and Diagnosis	180
Clock Overruns	181
7.3.7 Status Monitoring and Diagnostics	182



	Status Information Returned from the Scan System	182
7.4	Laser Control	183
7.4.1	Enabling, Activating and Switching Laser Control Signals	183
	Signals for "Laser Active" Operation	185
	Signals for "Laser Standby" Operation	185
	Automatic Suppression of Laser Control Signals	186
7.4.2	Configuring the LASER Connector	186
7.4.3	CO ₂ Mode	187
7.4.4	YAG Modes 1, 2, 3, 5	189
	Q-Switch Signal	189
	FirstPulseKiller Signal	189
	Differences Between the YAG Modes	189
	Lamp Current (Laser Power)	190
7.4.5	Laser Mode 4	192
7.4.6	Laser Mode 6	193
7.4.7	Softstart Mode (not yet implemented)	194
7.4.8	Pulse Picking Laser Mode	195
7.4.9	"Automatic Laser Control"	196
	Position-Dependent Laser Control	198
	Speed-Dependent Laser Control	200
	Encoder-Speed-Dependent Laser Control	203
	Loading and Determining the Nonlinearity Curve	203
	Vector-Defined Laser Control	205
7.4.10	Synchronization of the RTC6 Clock Cycle and an External Clock Signal	206
7.4.11	Pulse Synchronization Mode	207
7.5	Marking Dates, Times and Serial Numbers	209
7.5.1	Marking the Time and Date	209
7.5.2	Marking Serial Numbers	209
8	Advanced Functions for Scan Head Control and Laser Control	211
8.1	iDRIVE Functions	211
8.1.1	Transfer Protocol	211
8.1.2	Configuring the Data Signal Transmission Behavior of the Scan System	212
	Setting Data Types	212
	Reading Out Data	212
8.1.3	Monitoring the Positioning	213
8.1.4	Selecting the Tuning (Dynamics Setting)	215
8.1.5	Jump Mode	216
	Functional Principle	216
	Requirements and Activation	217
	Jump-Length-Dependent Jump Delays	217
	Determining Jump Delay Values Experimentally	218
	Notes on Loading Determined Jump Delay Values	218
	Automatic Determination of the Jump Delay Table	220
8.1.6	Configuring the PosAck Limit Value	221
8.1.7	Configuring the Effective Calibration	221
8.1.8	Configuring the Start Behavior	222
8.1.9	Fault Diagnosis and Functional Test	222
8.2	Coordinate Transformations	223
8.3	Online Positioning	227
8.3.1	"Local Online Positioning"	227
	Configuring "Local Online Positioning"	228
8.3.2	"Global Online Positioning"	230



Configuring "Global Online Positioning"	230
8.4 Wobbel Mode	231
Example Code (C++)	232
8.4.1 Wobbel Shapes – Important Notes on Choosing Appropriate Parameter Values	233
"Classic" Wobbel Shapes	233
"Freely Definable Wobbel Shapes"	234
8.5 Controlling 2D Scan Systems and 3D Scan Systems	235
8.5.1 2D Scan Systems	235
8.5.2 3D Scan Systems	236
Intended Use	236
Connection and Initialization	236
3D Commands	237
Adjusting Tracking Errors	238
Enhanced 3D Correction	239
8.5.3 Using Several Correction Tables	240
8.6 Processing-on-the-fly	241
8.6.1 Intended Use and Initialization	241
Overview	241
8.6.2 Compensating Linear Movements	242
Correction via Encoder Counters	242
Correction via McBSP Interface	244
Correction via McBSP Interface with Additional McBSP Input	245
Correction via McBSP Interface with Enhanced McBSP Input	245
8.6.3 Compensating Rotary Movements	246
Correction via Encoder Counter	246
Correction via McBSP Interface	247
Correction via McBSP Interface with Additional McBSP Input	247
8.6.4 Compensating 2D Motions	248
2D Encoder Compensation for xy Positioning Stages	248
8.6.5 Deactivating Processing-on-the-fly Correction	250
8.6.6 Virtual Image Field with Processing-on-the-fly	251
8.6.7 Synchronizing Processing-on-the-fly Applications	251
8.6.8 Encoder Resets	253
8.6.9 Monitoring Processing-on-the-fly Corrections	254
Customer-Defined Monitoring Area	255
8.6.10 Tracking Error Compensation of Encoder Values for Processing-on-the-fly Applications – NOT YET IMPLEMENTED	256
8.6.11 Processing-on-the-fly Correction for the z Axis	256
8.6.12 "Fly Extension" Commands	258
Notes on How to Use "Fly Extension" Commands	259
8.7 Pixel Output Mode	262
8.7.1 Principle of Operation	262
8.7.2 RTC6 Commands	262
8.7.3 Laser Control	264
8.7.4 Synchronization	267
8.8 micro_vector[*] Commands	272
8.9 Timed Commands	273
8.10 Automatic Self-Calibration	274
8.10.1 Use for Drift Compensation	274
8.10.2 How it Works	274
8.10.3 Determining Reference Values	276
8.10.4 Calibration During the Application	276



Automatic Self-Calibration	276
Customer-Specific Calibration	277
Supplemental Information about Calibration	277
8.11 Camming	278
8.12 Time Measurements	280
8.12.1 RTC6 Timer	280
8.12.2 Timestamps	280
9 Programming Peripheral Interfaces	281
9.1 Signal Output	281
9.1.1 16-Bit Digital Output Port	281
9.1.2 8-Bit Digital Output Port	282
9.1.3 2 Bit Digital Output Port	282
9.1.4 12-Bit Analog Output Port 1 and 2	282
9.1.5 Controlling Stepper Motors	283
Output Signals	283
Reference Movements and Position Initialization	284
Set-Position Movements	284
Querying Signals and Status Values	285
Terminating Infinite Movements	285
WaitTime Parameter	285
9.1.6 RS-232 Interface	286
9.1.7 McBSP Interface	286
9.2 Signal Input	287
9.2.1 16-Bit Digital Input Port	287
9.2.2 2-Bit Digital Input Port	287
9.2.3 RS-232 Interface	287
9.2.4 McBSP Interface	287
9.3 Control by External Signals	288
9.3.1 Starting and Stopping Lists by External Control Signals and Master/Slave Synchronization	288
External Stop	288
External Start	289
External Start with Track Delay	291
Regular (Periodic) External Starts	292
9.3.2 Conditional Command Execution	294
Example Code (Pascal)	295
9.3.3 Synchronization by Encoder Signals	297
Intended Use	297
Input Ports for External Encoder Signals	298
Encoder Simulation	298
9.3.4 Synchronization and Online Positioning by McBSP Signals	299
9.4 Periodical I/O Signals	300
10 RTC6 Commands	301
10.1 Overview	301
10.1.1 Designations	301
Multi-Board Commands	301
Normal, Short, Variable and Multiple List Commands	301
Undelayed Short List Commands, Delayed Short List Commands	302
Multiple List Commands	303
10.1.2 Compatibility	303
10.1.3 Version Information	304



10.1.4 Optional Functions	304
10.1.5 Control Commands	305
10.1.6 List Commands	309
10.1.7 Data Types	313
Pointer to Locations in the PC Memory	313
10.2 RTC6 Command Set	314
acquire_rtc	315
activate_fly_1_axis	317
activate_fly_2_axes	318
activate_fly_2d	320
activate_fly_2d_encoder	321
activate_fly_xy	323
activate_fly_xy_encoder	323
activate_scanahead_autodelays	324
activate_scanahead_autodelays_list	324
apply_mcbsp	325
apply_mcbsp_list	326
arc_abs	327
arc_abs_3d	328
arc_rel	329
arc_rel_3d	330
auto_cal	331
auto_change	334
auto_change_pos	335
bounce_supp	336
camming	337
clear_fly_overflow	339
clear_fly_overflow_ctrl	339
clear_io_cond_list	340
config_laser_signals	341
config_laser_signals_list	342
config_list	343
control_command	345
copy_dst_src	347
create_dat_file	348
disable_laser	349
enable_laser	350
eth_assign_card	351
eth_assign_card_ip	352
eth_boot_dcmd	353
eth_boot_timeout	353
eth_check_connection	354
eth_configure_link_loss	355
eth_convert_ip_to_string	356
eth_convert_string_to_ip	357
eth_count_cards	358
eth_found_cards	359
eth_get_card_info	360
eth_get_card_info_search	361
eth_get_com_timeouts	362
eth_get_com_timeouts_auto	363
eth_get_error	364

eth_get_ip	365
eth_get_ip_search	365
eth_get_last_error	366
eth_get_port_numbers	368
eth_get_serial_search	368
eth_get_standalone_status	369
eth_get_static_ip	370
eth_max_card	371
eth_remove_card	372
eth_search_cards	373
eth_search_cards_range	374
eth_set_com_timeouts	375
eth_set_com_timeouts_auto	376
eth_set_high_performance_mode	377
eth_set_port_numbers	378
eth_set_search_cards_timeout	379
eth_set_static_ip	380
execute_at_pointer	381
execute_list	382
execute_list_1	382
execute_list_2	382
execute_list_pos	383
fly_return	384
fly_return_1_axis	385
fly_return_2_axes	386
fly_return_3_axes	387
fly_return_z	388
free_rtc6_dll	389
get_auto_cal	390
get_bios_version	391
get_card_type	391
get_char_pointer	392
get_config_list	393
get_counts	393
get_dll_version	394
get_encoder	394
get_error	395
get_fly_2d_offset	398
get_free_variable	398
get_galvo_controls	399
get_head_para	401
get_head_status	402
get_hex_version	404
get_hi_data	404
get_hi_pos	405
get_input_pointer	406
get_io_status	406
get_jump_table	407
get_lap_time	407
get_laser_pin_in	408
get_last_error	408
get_list_pointer	409

get_list_serial	410
get_list_space	410
get_marking_info	411
get_master_slave	413
get_mcbsp	414
get_mcbsp_list	414
get_out_pointer	414
get_overrun	415
get_RTC_mode	416
get_RTC_version	417
get_scanahead_params	417
get_serial	418
get_serial_number	418
get_standby	419
get_startstop_info	420
get_status	422
get_stepper_status	424
get_sub_pointer	425
get_sync_status	426
get_table_para	428
get_temperature	428
get_text_table_pointer	429
get_time	429
get_timestamp_long	430
get_transform	431
get_value	434
get_values	436
get_wait_status	437
get_waveform	438
get_waveform_offset	439
get_z_distance	440
goto_xy	441
goto_xyz	442
home_position	443
home_position_xyz	444
if_cond	445
if_fly_x_overflow	445
if_fly_y_overflow	446
if_fly_z_overflow	447
if_not_activated	448
if_not_cond	449
if_not_fly_x_overflow	450
if_not_fly_y_overflow	451
if_not_fly_z_overflow	452
if_not_pin_cond	452
if_pin_cond	453
init_fly_2d	454
init_RTC6_dll	455
jump_abs	457
jump_abs_3d	458
jump_abs_drill	458
jump_abs_drill_2	458



jump_rel	459
jump_rel_3d	460
jump_rel_drill	460
jump_rel_drill_2	460
laser_on_list	461
laser_on_pulses_list	462
laser_signal_off	464
laser_signal_off_list	464
laser_signal_on	465
laser_signal_on_list	465
list_call	466
list_call_abs	468
list_call_abs_cond	469
list_call_abs_repeat	469
list_call_cond	470
list_call_repeat	471
list_continue	472
list_jump_cond	473
list_jump_pos	474
list_jump_pos_cond	475
list_jump_rel	476
list_jump_rel_cond	477
list_next	478
list_nop	478
list_repeat	479
list_return	480
list_until	481
load_auto_laser_control	482
load_char	484
load_correction_file	485
load_disk	489
load_fly_2d_table	492
load_jump_table	493
load_jump_table_offset	494
load_list	496
load_position_control	498
load_program_file	500
load_stretch_table	503
load_sub	505
load_text_table	506
load_varpolydelay	507
load_z_table	509
load_z_table_20b	510
load_z_table_no	511
load_z_table_no_20b	512
load_zoom_correction_file	513
long_delay	514
mark_abs	515
mark_abs_3d	516
mark_char	517
mark_char_abs	518
mark_date	519



mark_date_abs	521
mark_ellipse_abs	522
mark_ellipse_rel	523
mark_rel	524
mark_rel_3d	525
mark_serial	526
mark_serial_abs	528
mark_text	529
mark_text_abs	530
mark_time	531
mark_time_abs	533
master_slave_config	534
mcbsp_init	535
mcbsp_init_spi	536
measurement_status	537
micro_vector_abs	538
micro_vector_abs_3d	539
micro_vector_rel	540
micro_vector_rel_3d	541
move_to	541
number_of_correction_tables	542
para_jump_abs	543
para_jump_abs_3d	544
para_jump_rel	545
para_jump_rel_3d	546
para_laser_on_pulses_list	547
para_mark_abs	549
para_mark_abs_3d	551
para_mark_rel	552
para_mark_rel_3d	553
park_position	554
park_position_1_axis	556
park_position_2_axes	557
park_return	558
park_return_1_axis	560
park_return_2_axes	561
pause_list	562
periodic_toggle	563
periodic_toggle_list	564
quit_loop	565
range_checking	566
read_abc_from_file	568
read_abc_from_file_20b	569
read_analog_in	570
read_encoder	571
read_image_eth	572
read_io_port	573
read_io_port_buffer	574
read_io_port_list	575
read_mcbsp	576
read_multi_mcbsp	577
read_status	578

read_user_data	580
regulation3	581
release_RTC	582
release_wait	583
reset_error	584
restart_list	585
rs232_config	585
rs232_read_data	586
rs232_write_data	587
rs232_write_text	587
rs232_write_text_list	588
rtc6_count_cards	589
save_and_restart_timer	590
save_disk	591
select_char_set	593
select_cor_table	594
select_cor_table_list	597
select_RTC	598
select_serial_set	600
select_serial_set_list	600
send_user_data	601
set_angle	602
set_angle_list	603
set_auto_laser_control	604
set_auto_laser_params	608
set_auto_laser_params_list	608
set_char_pointer	609
set_char_table	610
set_control_mode	611
set_control_mode_list	613
set_default_pixel	614
set_default_pixel_list	614
set_defocus	615
set_defocus_list	616
set_defocus_offset	617
set_defocus_offset_list	618
set_delay_mode	619
set_delay_mode_list	620
set_DSP_mode	621
set_duty_cycle_table	622
set_ellipse	623
set_encoder_speed	624
set_encoder_speed_ctrl	625
set_end_of_list	626
set_eth_boot_control	627
set_extstartpos	628
set_extstartpos_list	628
set_ext_start_delay	629
set_ext_start_delay_list	630
set_firstpulse_killer	631
set_firstpulse_killer_list	631
set_fly_1_axis	632



set_fly_2_axes	633
set_fly_2d	635
set_fly_3_axes	637
set_fly_limits	639
set_fly_limits_z	640
set_fly_rot	641
set_fly_rot_pos	642
set_fly_tracking_error	643
set_fly_x	644
set_fly_x_pos	645
set_fly_y	646
set_fly_y_pos	647
set_fly_z	648
set_free_variable	649
set_free_variable_list	649
set_hi	650
set_input_pointer	651
set_io_cond_list	652
set_jump_mode	653
set_jump_mode_list	656
set_jump_speed	657
set_jump_speed_ctrl	657
set_jump_table	658
set_laser_control	659
set_laser_delays	662
set_laser_mode	663
set_laser_off_default	663
set_laser_pin_out	664
set_laser_pin_out_list	664
set_laser_power	665
set_laser_pulse_sync	666
set_laser_pulses	667
set_laser_pulses_ctrl	668
set_laser_timing	669
set_laser_timing_table	670
set_list_jump	671
set_mark_speed	672
set_mark_speed_ctrl	672
set_matrix	673
set_matrix_list	675
set_max_counts	676
set_mcbsp_freq	676
set_mcbsp_global_matrix	677
set_mcbsp_global_matrix_list	677
set_mcbsp_global_rot	678
set_mcbsp_global_rot_list	678
set_mcbsp_global_x	679
set_mcbsp_global_x_list	679
set_mcbsp_global_y	680
set_mcbsp_global_y_list	680
set_mcbsp_in	681
set_mcbsp_in_list	683

set_mcbsp_matrix	684
set_mcbsp_matrix_list	685
set_mcbsp_out	686
set_mcbsp_out_oie_ctrl	687
set_mcbsp_out_oie_list	687
set_mcbsp_out_ptr	688
set_mcbsp_out_ptr_list	689
set_mcbsp_rot	690
set_mcbsp_rot_list	690
set_mcbsp_x	691
set_mcbsp_x_list	691
set_mcbsp_y	692
set_mcbsp_y_list	692
set_multi_mcbsp_in	693
set_multi_mcbsp_in_list	695
set_n_pixel	696
set_offset	697
set_offset_list	697
set_offset_xyz	698
set_offset_xyz_list	699
set_pause_list_cond	700
set_pause_list_not_cond	701
set_pixel	702
set_pixel_line	703
set_pixel_line_3d	706
set_port_default	707
set_port_default_list	708
set_pulse_picking	709
set_pulse_picking_length	709
set_pulse_picking_list	710
set_qswitch_delay	710
set_qswitch_delay_list	710
set_rot_center	711
set_rot_center_list	711
set rtc4_mode	712
set rtc5_mode	713
set rtc6_mode	714
set_scale	715
set_scale_list	715
set_scanahead_laser_shifts	716
set_scanahead_laser_shifts_list	716
set_scanahead_line_params	716
set_scanahead_line_params_list	716
set_scanahead_params	716
set_scanahead_speed_control	716
set_scanner_delays	717
set_serial	717
set_serial_step	718
set_serial_step_list	718
set_sky_writing	719
set_sky_writing_limit	720
set_sky_writing_limit_list	720



set_sky_writing_list	721
set_sky_writing_mode	722
set_sky_writing_mode_list	723
set_sky_writing_para	724
set_sky_writing_para_list	726
set_softstart_level	727
set_softstart_level_list	727
set_softstart_mode	728
set_softstart_mode_list	728
set_standby	729
set_standby_list	730
set_start_list	731
set_start_list_1	731
set_start_list_2	731
set_start_list_pos	732
set_sub_pointer	733
set_text_table_pointer	734
set_timelag_compensation	735
set_trigger	736
set_trigger4	744
set_vector_control	745
set_verify	747
set_wait	748
set_wobbel	749
set_wobbel_control	751
set_wobbel_direction	753
set_wobbel_mode	754
set_wobbel_mode_phase	756
set_wobbel_offset	757
set_wobbel_vector	758
set_wobbel_vector_2	762
set_zoom	762
set_zoom_list	762
simulate_encoder	763
simulate_ext_start	764
simulate_ext_start_ctrl	765
simulate_ext_stop	766
spot_distance	767
spot_distance_ctrl	767
start_loop	768
stepper_abs	769
stepper_abs_list	770
stepper_abs_no	770
stepper_abs_no_list	771
stepper_control	772
stepper_control_list	772
stepper_disable_switch	773
stepper_enable	774
stepper_enable_list	774
stepper_init	775
stepper_rel	777
stepper_rel_list	777



stepper_rel_no	778
stepper_rel_no_list	778
stepper_wait	779
stop_execution	780
stop_list	780
stop_trigger	781
store_encoder	781
store_program	782
store_timestamp_counter	783
store_timestamp_counter_list	783
sub_call	784
sub_call_abs	785
sub_call_abs_cond	785
sub_call_abs_repeat	786
sub_call_cond	786
sub_call_repeat	787
switch_ioport	788
sync_slaves	789
time_control_eth	791
time_fix	792
time_fix_f	792
time_fix_f_off	793
time_update	794
timed_arc_abs	795
timed_arc_rel	796
timed_jump_abs	797
timed_jump_abs_3d	798
timed_jump_rel	799
timed_jump_rel_3d	800
timed_mark_abs	801
timed_mark_abs_3d	802
timed_mark_rel	803
timed_mark_rel_3d	804
timed_para_jump_abs	805
timed_para_jump_abs_3d	806
timed_para_jump_rel	807
timed_para_jump_rel_3d	808
timed_para_mark_abs	809
timed_para_mark_abs_3d	810
timed_para_mark_rel	811
timed_para_mark_rel_3d	812
transform	813
uart_config	816
upload_transform	817
verify_checksum	819
wait_for_1_axis	820
wait_for_2_axes	822
wait_for_encoder	824
wait_for_encoder_in_range	825
wait_for_encoder_in_range_mode	826
wait_for_encoder_mode	827
wait_for_mcbsp	829



wait_for_timestamp_counter	830
wait_for_timestamp_counter_long	831
wait_for_timestamp_counter_mode	832
write_8bit_port	833
write_8bit_port_list	833
write_abc_to_file	834
write_abc_to_file_20b	835
write_da_1	836
write_da_1_list	836
write_da_2	837
write_da_2_list	837
write_da_x	838
write_da_x_list	839
write_hi_pos	840
write_image_eth	841
write_io_port	842
write_io_port_list	842
write_io_port_mask	843
write_io_port_mask_list	843
Undelayed Short List Command	844
write_port_list	844
10.3 Unsupported RTC4/RTC5 Commands	846
11 Demo Programs	848
12 Troubleshooting	849
13 Customer Service	851
13.1 Servicing and Repairs	851
13.2 Warranty	851
13.3 Contacting SCANLAB	851
13.4 Product Disposal	851
14 Legal	852
14.1 EU Declaration of Conformity – RTC6 PCIe Board	852
14.2 Compliance with FCC Rules	853
15 Technical Specifications – RTC6 PCIe Board	854
16 Appendix A: The RTC6 Ethernet Board	858
16.1 Product Overview	858
16.1.1 RTC6 Ethernet Board vs. RTC6 PCIe Board – Usage and Comparison	858
16.1.2 System Requirements	859
Hardware	859
Software	859
16.1.3 Options	860
16.1.4 Labeling	860
16.1.5 Type Identification	860
16.1.6 Unpacking Instructions and Typical Scope of Delivery	860
16.1.7 Delivered Software	860
16.1.8 Accessories for the RTC6 PCIe Board	860
16.1.9 Supplementary Software	860
16.2 Layout, Interfaces, Jumper Settings	861
16.2.1 Layout – Upper Side	861
16.2.2 Layout – Lower Side	862
16.2.3 Dimensions and Connector Positions	863



16.2.4 LASER Socket Connector	864
Laser Control Signals	864
External Control Signals	864
BUSY List Execution Status	864
2-Bit Digital Input Port and 2-Bit Digital Output Port	864
12-Bit Analog Output Ports	865
Input and Output Wiring	865
16.2.5 SCANHEADs Socket Connector	867
16.2.6 POWER Socket Connector	869
16.2.7 ETH Connector	869
16.2.8 SPI/ANA/UART Socket Connector	870
Analog Input Ports	870
McBSP Interface	870
RS-232 Interface	870
16.2.9 STEPPER Socket Connector	871
16.2.10MOF Socket Connector	871
Encoder Input Ports	872
External Control Signals	872
BUSY OUT Status	872
16.2.11EXTN. 1 Socket Connector	872
Configuring the Output Signal Level	873
16-Bit Digital Output Port and 16-Bit Digital Input Port	873
Synchronization of Data Acquisition	873
BUSY List Execution Status	873
16.2.12EXT. 2 Socket Connector	873
Configuration by Solder Jumpers	874
8-Bit Digital Output Port	874
16.2.13Master Socket Connector, Slave Socket Connector	875
16.2.14Jumper Settings	876
Solder Jumper Field A – Configuring the Output Signal Level at EXTENSION 1 Socket Connector	876
Solder Jumper Field B – Configuring pin (09) of EXT. 2 Socket Connector	877
Solder Jumper Field C – Configuring pin (08) of EXT. 2 Socket Connector	878
Jumper Field 'Force DHCP'	879
16.2.15Real-Time Clock	879
16.3 Installation and Operation	880
16.3.1 Hardware Installation	880
16.3.2 Software Installation	881
16.3.3 Connecting to a Network	881
16.4 Notes on Migrating Existing and Programming New RTC6 User Programs	882
16.4.1 Finding RTC6 Ethernet Boards in the Network and Querying their Properties	882
16.4.2 Example Code (C++): Initialization Covering RTC6 Ethernet Boards	883
16.5 RTC6 Ethernet Board Commands and Functions	886
16.5.1 Notes on Working with IP Addresses	886
16.5.2 About Searching RTC6 Ethernet Boards	886
16.5.3 About the RTC6 Board Management	887
16.5.4 Checking the Connection to the RTC6 Ethernet Board	888
16.5.5 Command Set for the RTC6 Ethernet Board	888
16.6 Safe Startup and Shutdown Sequences	889
16.7 Standalone Functionality	890
16.7.1 Upgrading BIOS-ETH	891
16.7.2 Preparing the "Standalone Basic State"	891



16.7.3 Preparing the "Standalone Full State"	892
16.7.4 Boot Image	893
Creating a Boot Image on the PC	893
Copying Boot Image to Board(s)	893
Procedure after an Transmission Abortion	893
16.7.5 Control Commands Allowed for Automatic Booting	894
16.7.6 Automatic Booting – Process in Detail	896
16.8 "High Performance Mode"	897
16.9 Legal	898
16.9.1 EU Declaration of Conformity – RTC6 Ethernet Board	898
16.9.2 Compliance with FCC Rules	899
16.9.3 TI-RTOS	899
16.10 Technical Specifications – RTC6 Ethernet Board	900
17 Appendix B: The UFPM Extension Board	905
18 Appendix C: The RTC6 EtherBox	908
18.1 Mounting	911
18.2 Cabling	911
18.3 Installation and Operation	912
18.4 Legal	913
18.4.1 EU Declaration of Conformity – RTC6 EtherBox	913
18.4.2 Compliance with FCC Rules	914
18.5 Technical Specifications – RTC6 EtherBox	914
19 Appendix D: RTC6 Software Packages – History	915
20 Appendix E: iDRIVE Scan Systems – Control Commands and Signals Transmitted to RTC Control Boards	916
21 Appendix G: Change Index	948



1 About this Manual

This manual describes the available SCANLAB RTC6 boards and their usage for synchronous control of scan systems, lasers and peripheral equipment.

The main chapters of this manual use the RTC6 PCIe Board (= "RTC6 PCI Express Board") to exemplify. For a product overview, see [Chapter 2 "Product Overview", page 30](#).

Other RTC6 board variants are described in the Appendix:

- RTC6 Ethernet Board, see [Chapter 16 "Appendix A: The RTC6 Ethernet Board", page 858](#)

The manual is a part of the product. Read these instructions carefully before you proceed with installing and operating.

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.

This manual refers to:

- [RTC6 Software Package V1.14.1](#)

DLL file for 32 bit user programs ^(a)	RTC6DLL.dll	DLL 634^(b)
DLL file for 64 bit user programs ^(a)	RTC6DLLx64.dll	
Program file for the PCIe-DSP	RTC6OUT.out	OUT 635^(b)
Program file for the Eth-DSP	RTC6ETH.out	ETH 635^(b)
Firmware file for the FPGA	RTC6RBF.rbf	RBF 634^(b)
Auxiliary file	RTC6DAT.dat	DAT 604^(b)

(a) Software for laser-scan processes, which controls RTC6 boards based on this [RTC6 DLL](#) file is consistently denoted as "user program" in this manual.

(b) Abbreviated version in this manual for the Versions.

The version numbers of the supplied [RTC6 DLL](#) and [RTC6 files](#) are indicated in the names of the corresponding [zip](#) files, see [Section "Folder RTC6 Files", page 32](#).

To identify the version numbers of your files after installation, use [get_dll_version](#), [get_hex_version](#) and [get_RTC_version](#).

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 Related Documents

- "excelliSCAN Scan Heads – Functional Principle of SCANAhead Servo Control and Operation by RTC6 Boards" Manual
- "Calibrating a 3-Axis Laser Scan System" Manual



1.3 Glossary and Abbreviations

[*]mark[*] Command	All commands with "mark" as part of their names. See Section "Mark Commands", page 136 .
[*]para[*] Command	All commands with "para" as part of their names. See Section "[*]Para[*] Commands", page 138 .
3D image field	Synonym: working volume (process volume). See Chapter 7.3.4 "3D Image Field", page 169 .
"Arc" command	Umbrella term for Arc Commands and Ellipse Commands .
BCD	Binary Coded Decimal.
BIOS	Basic Input/Output System. Is permanently stored in the Flash memory of the RTC6 board. See also RTC6conf.exe .
BUSY pin	Pin with BUSY OUT signal (= BUSY list execution status). <ul style="list-style-type: none"> • RTC6 PCIe Board: <ul style="list-style-type: none"> – LASER Connector, page 72 – EXTENSION 1 Socket Connector, page 77 • RTC6 Ethernet Board: <ul style="list-style-type: none"> – LASER Socket Connector, page 864 – EXTENSION 1 Socket Connector, page 872
DSP	Digital signal processor on the RTC6 board.
Dynamic focusing unit	This includes, for example, the following SCANLAB products: varioSCAN, varioSCAN _{de} , varioSCAN FC and varioSCAN FLEX, excelliSHIFT.
Ethernet Link Loss	The cable connection to the RTC6 Ethernet Board has been disconnected.
Flash memory	Non-volatile memory on the RTC6 board that replaces the EEPROM of the RTC5 board.
FPGA	Field programmable gate array on the RTC6 board.
Hardware reset	New start after powering the RTC6 board. Synonym: "power up", "power cycle".
Hard jump	Direct output to a specified position. Decomposition into Microsteps into a single 10 µs clock cycle.
iDRIVE scan systems	In this manual, the term subsumes, for example, the following SCANLAB products: intelliSCAN, intelliSCAN _{de} , intelliSCAN _{se} , intelliDRILL, intellicube, intelliWELD, varioSCAN _{de} , powerSCAN II 50i, excelliSCAN.
Image field	Synonym: Working field .



intelliSCAN	In this manual, the term subsumes, for example, the following SCANLAB products: intelliSCAN, intelliSCAN _{de} , intelliSCAN _{se} , intelliDRILL, intellicube, intelliWELD, powerSCAN II 50i.
Jump command	Serves to move the scan system axes to a new position while the laser is <i>off</i> . See also Section "Jump Commands", page 136 .
	See 2D Jump Commands, page 309 and 3D Jump Commands, page 309 .
LSB	Least Significant Bit.
Mark command	Serves to perform marking motions while the laser is switched <i>on</i> . Examples: mark, arc and ellipse. See 2D Mark Commands, page 309 , and 3D Mark Commands⁽¹⁾, page 309 .
McBSP	Multi channel Buffered Serial Port.
Microstep	See Chapter 7.1.2 "Microstepping", page 139 .
Microvector	The term refers to micro_vector[*] commands , see Chapter 8.8 "micro_vector[*] Commands", page 272 .
micro_vector[*] command	All commands with "micro_vector" as part of their names. See Chapter 8.8 "micro_vector[*] Commands", page 272 .
MSB	Most Significant Bit.
NAND memory	Non-volatile memory on the RTC6 Ethernet Board. Synonym: NAND flash.
NULL	Means on the one hand the number 0, on the other hand a pointer with the value 0. The spelling for this is different in the different programming languages.
OIE	Open Interface Extension. SCANLAB product (hardware). See SCANLAB website.
PCB	Printed Circuit Board.
PID	Proportional–Integral–Derivative. Category of controller.
Pixel mode	Brief for "Pixel Output Mode". See Chapter 8.7 "Pixel Output Mode", page 262 .
Polyline	A direct sequence of [*]mark[*] Commands or "Arc" commands . The marking is continuous. Synonym: polygonal chain, polygonal line, polygonal traversal.
PosAck	"Position Acknowledge", see PosAck signal .



PosAck signal	<ul style="list-style-type: none">As of scan system firmware \geq 2001.PosAck signal-capable SCANLAB scan systems, for example, intelliSCANRefers to the meaning of:<ul style="list-style-type: none">Bit #12, Bit #4, Bit #11, Bit #3 of the XY2-100 status word transferred with 16-bit protocolBit #16, Bit #8, Bit #15, Bit #7 of the XY2-100 status word transferred with 20-bit protocolThese bits are set, if the position errors of x axis and y axis are in the allowed range:<ul style="list-style-type: none">See also "excelliSCAN Scan Heads – Functional Principle of SCANAhead Servo Control and Operation by RTC6 Boards" Manual, Chapter 3.1.3 "Returned Data Signal Differences: Status Signal Behavior after Limit Value Exceedance", page 14.For more information, refer to the corresponding scan head manuals.Compare to TrAck signal.
Processing-on-the-fly session	A section of a list that uses external inputs (encoder pulses, McBSP transmissions) to correct moving workpiece positions.
Reset of the RTC6 board	Synonym: Software reset .
SCANAhead system	SCANLAB scan system with SCANAhead servo control, for example, scan heads of the excelliSCAN series. For further information see " excelliSCAN Scan Heads – Functional Principle of SCANAhead Servo Control and Operation by RTC6 Boards " Manual.
Software reset	Restart after load_program_file . This does not reset everything, for example, loaded correction tables are retained. Synonym: Reset of the RTC6 board .
SPI	Serial Peripheral Interface.
Standalone Operation Mode	See Chapter 16.7 "Standalone Functionality" , page 890.
TrAck	"Trajectory Acknowledge", see TrAck signal .



TrAck signal	<ul style="list-style-type: none">As of scan system firmware $\geq 5102 + \geq 5112$.TrAck signal-capable SCANLAB scan systems, for example, SCANAhead systemsRefers to the meaning of:<ul style="list-style-type: none">Bit #12, Bit #4, Bit #11, Bit #3 of the XY2-100 status word transferred with 16-bit protocolBit #16, Bit #8, Bit #15, Bit #7 of the XY2-100 status word transferred with 20-bit protocolThese bits are set, if the Trajectory errors of x axis and y axis are in the allowed range:<ul style="list-style-type: none">See also "excelliSCAN Scan Heads – Functional Principle of SCANAhead Servo Control and Operation by RTC6 Boards" Manual, Chapter 3.1.3 "Returned Data Signal Differences: Status Signal Behavior after Limit Value Exceedance", page 14.For more information, refer to the corresponding scan head manuals.Compare to PosAck signal.
Trajectory	In this manual: curve with $10 \mu s$ parameterization.
Trajectory error	Deviation of the actual Trajectory from the set Trajectory .
Tracking error	Time difference between the planned and actual reaching of a certain mirror position at constant speed.
Vector command	Umbrella term for Jump Commands and Mark Commands .
Working field	Synonym: Image field .



2 Product Overview

2.1 Labeling

The serial number of the RTC6 PCIe Board is printed on a label attached to the board.

The article number and configuration of the board are described in the packaging list, see [Chapter 2.6 "Options", page 37](#), and [Chapter 2.7 "Jumper Settings and Type Designations", page 39](#).

2.2 Unpacking Instructions and Typical Scope of Delivery

- (1) Carefully remove the RTC6 PCIe Board from the package.
- (2) Keep the packaging, including the antistatic bag the RTC6 PCIe Board is delivered in, so that in case of repair the board can be properly repackaged and returned to SCANLAB.
- (3) Also remove all other articles from the package. Check that all parts have been delivered. Refer to the corresponding packaging list.
The scope of delivery typically includes an RTC6 PCIe Board and a data CD (with the RTC6 Software Package, see below, and this manual). Possibly additional components are also contained, see [Chapter 2.8 "Accessories for the RTC6 PCIe Board", page 43](#).

2.3 Delivered RTC6 Software Package

- See also [Chapter 19 "Appendix D: RTC6 Software Packages – History", page 915](#).

The delivered RTC6 Software Package contains the RTC6 board driver⁽¹⁾ for the 32-bit and 64-bit versions of the operating systems Microsoft Windows 10, 8, 7.

Windows XP and Windows Vista are not supported!

Observe the safety notice on legacy RTC board drivers in [Chapter 2.5.2 "Software", page 36](#).

The data CD contains all files as unzipped versions. The complete RTC6 Software Package is also delivered zipped for easy identification and management of different software versions:

- RTC6_Software_Package_Rev.n.n.n_<Datum>.zip

The content of the RTC6 Software Package is as follows:

- Readme.txt
Description in English
- Liesmich.txt
Description in German
- IMPORTANT_RELEASE_NOTES.txt
Bilingual English and German
- RTC6_Release_Notes_YYYY_MM_DD.pdf
Mentions known restrictions.
Bilingual English and German
- Folder Correction Files
- Folder CorrectionFileConverter
- Folder Demo Files
- Folder HPGL
- Folder iSCANcfg
- Folder RTC6 Driver
- Folder RTC6 Files
- Folder RTC6 Tools

(1) The RTC6 board driver is not required for RTC6 Ethernet Boards.



Folder Correction Files

- Cor_1to1.ct5
1to1 correction file⁽¹⁾⁽²⁾

Folder CorrectionFileConverter

- CorrectionFileConverter.exe
This Win32-based program converts RTC4 correction files *.ctb to RTC5/RTC6 correction files *.ct5 and vice versa. The corresponding manual is supplied in English and German.

Folder Demo Files

Subfolder Remote Interface

- telegrams.h
 - Subfolder example1
 - example_1.cpp
 - remote_example.cpp
 - remote_example.h
 - Subfolder example2
 - example_2.cpp

Folder HPGL

- Hpgl.exe is a Win32-based HPGL-to-RTC6 converter. See also [Chapter 5.6 "Functionality Test"](#), page 90.
- The folder contains some *.plt files (Hewlett Packard HPGL format (vector graphic plotter files) for test purposes.
- Hpgl.exe needs [RTC6 files](#) and the [RTC6DLL.dll](#) in the same folder.

Folder iSCANcfg

- iSCANcfg.exe is a Win32-based diagnosis and configuration program for [iDRIVE](#) scan systems⁽³⁾. RTC4, RTC5 and RTC6 (PCI/PCIe as well as Ethernet variants) are supported.
- Needs [RTC6 files](#) and the [RTC6DLL.dll](#) in the same folder and in addition RtcHalDLL.dll. The corresponding manual is supplied in English ([Manual_iSCANcfg_v1-7.pdf](#)) and German ([Handbuch_iSCANcfg_v1-7.pdf](#)).

Folder RTC6 Driver

(RTC6 board driver for Windows)

- RTC6DRV.sys, RTC6DRVx64.sys, RTC6DRVx86.sys
RTC6 board driver files
- RTC6DRV.inf
Installation file (setup information)
- RTC6DRVx64.cat, RTC6DRVx86.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

(1) Additional correction files (D2_XXX.CT5, D3_YYY.CT5) and *_ReadMe.txt file(s) are not part of the RTC6 Software Package.

(2) See also [Section "1to1 Correction Tables"](#), page 176.

(3) See Glossary entry on [page 26](#).

Folder RTC6 Files

- RTC6_Software_RevisionHistory_<Date>_<Rev>.pdf
Description of RTC6 Software Package changes in English
- RTC6_Software_Aenderungshistorie_<Date>_<Rev>.pdf
Description of RTC6 Software Package changes in German
- Subfolder Linux
- Subfolder Windows
- Subfolder Program Files

Subfolder Linux

- For Linux developers only:
Debian packages with shared libraries for controlling RTC6 Ethernet Boards
 - readme.txt
Package contents and installation instructions
 - Subfolder debian-stretch
Package for Debian Stretch
 - Subfolder debian-buster
Package for Debian Buster
 - Subfolder debian-bullseye
Package for Debian Bullseye

Subfolder Program Files

- RTC6 files
 - RTC6OUT.out
Program file for the PCIe-DSP
 - RTC6ETH.out
Program file for the Eth-DSP
 - RTC6RBF.rbf
Firmware file for the FPGA
 - RTC6DAT.dat
Binary auxiliary file
- Differing versions of **RTC6 files** and **RTC6 DLL** cannot be arbitrarily combined with another. The following **zip** files (each including a text file with version/compatibility information) are provided to make it easier to identify the versions:
 - RTC6OUT_<current OUT version number>.zip
(includes **RTC6OUT.out** and **RTC6ETH.out**)
 - RTC6RBF_<current RBF version number>.zip
 - RTC6DAT_<current DAT version number>.zip

Subfolder Windows

- RTC6 DLL
 - RTC6DLL.dll
Win32-based RTC6 dynamic link library
 - RTC6DLLx64.dll
Win64-based RTC6 dynamic link library
- Utility files for C, C++ and C#
 - RTC6expl.c
C functions for **RTC6 DLL** handling for explicit linking
 - RTC6expl.h
C function prototypes of the RTC6 for explicit linking of the **RTC6 DLL**
 - RTC6DLL.lib
Visual C++ import libraries for implicit linking of the **RTC6 DLL** for Win32-based applications
 - RTC6DLLx64.lib
Visual C++ import libraries for implicit linking of the **RTC6 DLL** for Win64-based applications
 - RTC6impl.h
C function prototypes of the RTC6 for implicit linking of the **RTC6 DLL**
 - RTC6impl.hpp
C++ function prototypes of the RTC6 for implicit linking of the **RTC6 DLL**
 - RTC6Wrap.cs
Import declarations of the wrapper class for implicit linking in C#
- Utility file for Delphi
 - RTC6Import.pas
Import declarations for Delphi
- Differing versions of **RTC6 files** and **RTC6 DLL** cannot be arbitrarily combined with another. The following **zip** file (including a text file with version/compatibility information) is provided to make it easier to identify the versions:
 - RTC6DLL_<current DLL version number>.zip
(includes DLLs and related utility files)



Folder RTC6 Tools

- RTC6conf.exe
"RTC6 Configuration Tool"
with the following main features
 - for RTC6 PCIe Boards and RTC6 Ethernet Boards: viewing board information (serial number, enabled options, **BIOS** version), enabling options (requires an auxiliary file which is to be purchased from **SCANLAB**), upgrading **BIOS**
 - for RTC6 Ethernet Boards additionally: searching boards in (configurable) network segments (subnets), show and change network settings for static IP configuration.
- RTC6conf.pdf
Description of the use of RTC6conf.exe
- RTC6BIOSOUT_23.out
DSP program file for RTC6 PCIe Board BIOS (version 0x23)
- RTC6BIOSETH_35.out
DSP program file for RTC6 Ethernet Board BIOS (version 0x35)⁽¹⁾
- The **RTC6 files** and **RTC6 DLL** of this RTC6 Software Package
 - RTC6DAT.dat
 - RTC6DLL.dll
 - RTC6ETH.out
 - RTC6OUT.out
 - RTC6RBF.rbf
- SleepMode.cmd
Script to deactivate all Windows sleep and hibernate modes.
- ReadMe_SleepMode.pdf
Description of the use of SleepMode.cmd

(1) See also **Chapter 16.7.1 "Upgrading BIOS-ETH", page 891.**

2.4 Intended Use

The SCANLAB RTC6 PCIe Board and its associated RTC6 Software Package is intended for synchronous real-time control of scan systems, lasers and peripheral equipment by a Windows PC with a PCIe bus interface.

RTC6 boards are available in different hardware variants, [Chapter 1 "About this Manual", page 24](#).

The delivered [RTC6 DLL](#) provides an extensive command set for control. This allows a quick and flexible software development for laser-scan processes.

The RTC6 PCIe Board is equipped with a fast digital signal processor ([DSP](#)). During execution of control commands it also handles more complex signal processing, such as simultaneous control of two scan systems or coordinate transformations.

Moreover, you can store controlling commands (= list commands) on the RTC6 PCIe Board and start their execution at a later point in time. Command execution by the RTC6 PCIe Board 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.

With the RTC6 PCIe Board, commonly used laser types can be controlled. To control lasers, the supplies interfaces that output laser control signals and are software-configurable for each application's requirements.

Users can choose among different laser modes and set the signal parameters (for example, the signal level – active-HIGH or active-LOW) or the output frequency to a suitable value.

For controlling peripheral equipment and incorporating external control signals, the RTC6 PCIe Board provides a range of interfaces (for example, a 16-bit digital input port, a 16-bit digital output port, two 12-bit analog output ports and an RS-232 interface, see [Chapter 4 "RTC6 PCIe Board – Layout and Interfaces", page 63](#)) and associated software commands.

As many RTC6 PCIe Board as the PCIe bus permits can be operated simultaneously in a single PC. The total number of RTC6 PCIe Boards and RTC6 Ethernet Boards must not exceed 255.

Moreover, the [RTC6 DLL](#) allows multi-threading as well as multi-processing; therefore several user programs can be used simultaneously.

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 [RTC6 DLL](#) serializes these accesses automatically.

The RTC6 PCIe Board is available in various configurations, see [Chapter 2.6 "Options", page 37](#), and [Chapter 2.7 "Jumper Settings and Type Designations", page 39](#).

The RTC6 PCIe Board interfaces are described on [Chapter 4 "RTC6 PCIe Board – Layout and Interfaces", page 63](#), installation and start-up on [Chapter 5 "Installation and Start-Up", page 87](#), and programming on [Chapter 6 "Developing RTC6-User Programs", page 92](#). Individual command descriptions are listed beginning with [Chapter 10 "RTC6 Commands", page 301](#).

The technical specifications of the RTC6 PCIe Board are summarized on [Chapter 15 "Technical Specifications – RTC6 PCIe Board", page 854](#).



Caution!

- Do not operate the RTC6 PCIe Board outside of the PC.
- The RTC6 PCIe 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 RTC6 PCIe 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 (for example, 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 RTC6 PCIe 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.

2.5 System Requirements

2.5.1 Hardware

The RTC6 PCIe Board requires a Windows PC with a PCIe bus interface and at least one free PCIe slot.

RTC6 PCIe Boards intended for synchronized master/slave operation should (recommended) be installed in adjacent PCIe slots.

2.5.2 Software

To operate the RTC6 PCIe Board, RTC6 board driver and **RTC6 DLL** files for Microsoft Windows operating systems must be used. These are included in the scope of delivery (RTC6 Software Package). For the supported Windows versions, see [Chapter 2.3 "Delivered RTC6 Software Package", page 30](#).

The RTC6 board driver supports the plug and play capability of the RTC6 PCIe Board as well as the simultaneous operation of up to 255 RTC6 PCIe Boards. The **RTC6 DLL** files contain the command set to control laser scan systems.

Notice!

- The RTC6 PCIe Board does not support power-saving modes, that switch off power to the PCIe bus. Accordingly, you must disable standby or sleep modes of the operating system. See also following [Section "Notes", page 36](#).

Notice!

- If on your PC an WDM technology-based RTC3/4/5 board driver
 - is yet installed or
 - was installed and has been removed or
 - you are not sure in this regard:
 - (1)Install the RTC6 board driver.
 - (2)Run `ScanlabClassChecker.cmd` as Administrator (part of the delivered RTC6 Software Package; see there also the background information in [ReadMe_ScanlabClassChecker.pdf](#)).
- Step 2 can be skipped on brand new PCs on which an RTC board driver never has been installed.

Notes

- RTC6 Software Packages only contain WDF⁽¹⁾ drivers (version 6.1.7600.16385). A WDM⁽²⁾ driver (outdated today) is not contained.
- If `init_rtc6_dll` is called, then the RTC6 board driver prevents automatic activation of standby or sleep modes (continuously until the next system restart). This enables RTC6 PCIe Boards to continue processing lists autonomously even when the initiating user program has already been terminated. However, standby or sleep modes cannot be prevented if triggered manually or by discharged batteries. Afterward, loaded lists and other settings of the RTC6 PCIe Board are lost. After a wake-up, the RTC6 PCIe Board might no longer be correctly addressable. Users themselves must prevent standby or sleep modes prior to the first call of `init_rtc6_dll`. Before calling `init_rtc6_dll` for the first time, make sure that standby or sleep modes of the operating system are deactivated. The script `SleepMode.cmd` which is contained in the RTC6 Software Package (under RTC6 Tools) deactivates *all* sleep and hibernation modes, see [ReadMe_SleepMode.pdf](#).

(1) Windows Driver Framework. Current technology.

(2) Windows Driver Model. Legacy technology.

2.6 Options

RTC6 PCIe Boards can be equipped with different options (features). For new boards, the options ordered are enabled/installed at the SCANLAB factory on delivery. For information on retrofitting already delivered boards, see [Section "Notes", page 38](#).

To query which options are actually enabled on a certain board, [get_RTC_version](#) is used.

The following options are available for RTC6 PCIe Boards:

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

Allows that a Processing-on-the-fly correction can be activated, see [Chapter 8.6 "Processing-on-the-fly", page 241](#).

- **Option "3D"**

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

Allows that an RTC6 PCIe Board can synchronously control a third axis (z axis, for example, a varioSCAN as a dynamic focus unit) along with the scan system's x axis and y axis (usually two galvanometer scanners) by its two scan head connectors, see [Chapter 8.5.2 "3D Scan Systems", page 236](#).

- **Option "Second Scan Head Control"**

- **Controlling Two Scan Systems Simultaneously**

Allows that an RTC6 PCIe Board can simultaneously control two xy-scan systems via its two scan head connectors, see also [Chapter 4.5.1 "Scan Head Connectors and Transfer Protocol", page 66](#) and [Chapter 8.5 "Controlling 2D Scan Systems and 3D Scan Systems", page 235](#).

- **Option "SCANa"**

- **Controlling Scan Systems with SCANAhead Servo Control**

Allows that scan systems equipped with SCANAhead servo control (for example, excelliSCAN series) can (also) be controlled. For further information, see the ["excelliSCAN Scan Heads – Functional Principle of SCANAhead Servo Control and Operation by RTC6 Boards" Manual](#).

- **Option "UFPM"**

- **Pixel Output Modes with pixel output frequencies more than 800 kHz⁽¹⁾**

Allows that pixel output frequencies of > 800 kHz...3,2 MHz can be achieved in certain [Pixel Output Modes](#), see also [set_pixel_line](#).

- **Option "syncA"**

- **Support of the syncAXIS control Software**

Allows that SCANLAB syncAXIS control software can be used (board and software are part of SCANLAB's scope of delivery for XL SCAN systems).

- **Option "DC/DC Converter"**

These boards are equipped with an extra DC/DC converter (optoelectronic coupler) ex works. Therefore, the laser control signals LASERON, LASER1 and LASER2 at the [LASER Connector](#) and EXTENSION 2 socket connector are galvanically decoupled from the PC ground⁽²⁾, see [Section "Laser Control Signals", page 72](#), and [Section "Laser Control Signals", page 80](#).

(1) UFPM, Ultra Fast Pixel Mode.

(2) With RTC6 Ethernet Boards: ground at POWER connector.



Notes

- Each RTC6 PCIe Board article number indicates which of the options above are enabled and/or installed. These are stated in the packing list. The naming there is as follows:
 - “Fly” for [Option Processing-on-the-fly](#)
 - “3D” for [Option “3D”](#)
 - “SSHC” for [Option “Second Scan Head Control”](#)
 - “SCANa” for [Option “SCANa”](#)
 - “UFPM” for [Option “UFPM”](#)
 - “syncA” for [Option “syncA”](#)
 - “DCDC” for [Option “DC/DC Converter”](#)
- To query which options are actually enabled on a board, [get_RTC_version](#) is used.
- If you need one or more options which are not activated out of the factory on your RTC6 PCIe Board: order an corresponding license file from SCANLAB, specifying the serial number and the options you want. Then, to apply the upgrade to the board, specify it in [RTC6conf.exe](#).
- For retrofitting your RTC6 PCIe Board with the extra DC/DC converter (optoelectronic coupler) you need to send it to SCANLAB.
- For optical data transfer between the RTC6 PCIe Board and the scan system, solely a suitable data cable is required, see [Chapter 4.5.3 “Data Cables \(Accessories\)”, page 70](#). Neither the RTC6 PCIe Board side nor the scan system side requires a special optical data interface for that.
- With software package version \leq [1.4.4](#), the “Processing-on-the-fly” functionality cannot be used for scan systems with [SCANahead control](#).

2.7 Jumper Settings and Type Designations

SCANLAB ships RTC6 PCIe Boards in various jumper configurations. Jumpers are connections which are either open or closed. The *factory* solder jumper configuration of an RTC6 PCIe Board can be identified by its article number.

In addition, a three-digit type code scheme is used, for example,

- “RTC6 PCI-Express TYPE 000”
 - No signals (that is, all solder jumpers are open)
- “RTC6 PCI-Express TYPE 124”
 - 5 V output signal level at the EXTENSION 1 socket connector
 - DATA7 at pin 15 of the EXTENSION 2 socket connector
 - LATCH at pin 17 of the EXTENSION 2 socket connector

Digit 1	Refers to the output signal level at the EXTENSION 1 socket connector ⁽¹⁾ .
	=0: No signals. =1: 5 V. =2: 3,3 V.
Digit 2	Refers to pin 15 of the EXTENSION 2 socket connector ⁽³⁾ . =0: No signals. =1: +5 V. =2: DATA7. =3: GROUND.
Digit 3	Refers to pin 17 of the EXTENSION 2 socket connector ⁽²⁾ . =0: No signals. =1: +5 V. =2: DATA7. =3: GROUND. =4: LATCH.

Trained users can reconfigure solder jumpers by using a soldering iron. The assignment of a desired signal is done by closing or opening (applying or removing solder or zero-ohm resistor) of corresponding solder jumpers as described in the following⁽¹⁾⁽²⁾⁽³⁾.

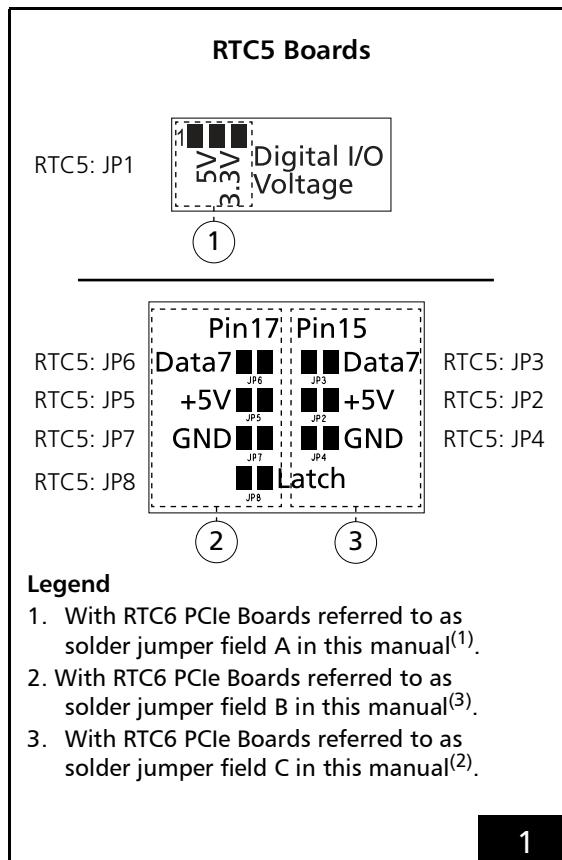
Notice!

- Only configure allowed jumper settings. Otherwise, the board gets damaged!

- (1) For the solder jumper setting, see [Chapter 2.7.1 “Solder Jumper Field A – Configuring the Output Signal Level at EXTENSION 1 Socket Connector”, page 40.](#)
- (2) For the solder jumper setting, see [Chapter 2.7.2 “Solder Jumper Field B – Configuring Pin \(17\) of EXTENSION 2 Socket Connector”, page 41.](#)
- (3) For the solder jumper setting, see [Chapter 2.7.3 “Solder Jumper Field C – Configuring Pin \(15\) of EXTENSION 2 Socket Connector”, page 42.](#)

Notes for RTC5 Users

- In contrast to RTC5 boards, jumper numbers are no longer imprinted on RTC6 PCIe Boards. Therefore, RTC5 jumper designations cannot longer be used in this RTC6 Manual. The relation of the designations establishes [Figure 1](#).



Jumper designations with RTC5 boards and RTC6 PCIe Boards.
Figure shows RTC5 board.

2.7.1 Solder Jumper Field A – Configuring the Output Signal Level at EXTENSION 1 Socket Connector

The solder jumper field A is located on the lower side of the RTC6 PCIe Board, see [Figure 6](#).

It is used to set the level (5 V or 3.3 V) of all output signals at the EXTENSION 1 socket connector, see the following table.

See also [Section "Configuring the Output Signal Level", page 77](#).

Allowed jumper setting	EXTENSION 1 socket connector
* closed open	Output signal level 5 V.
* open closed	Output signal level 3.3 V.
open open	No signal output.

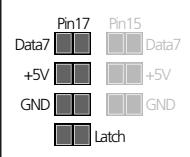
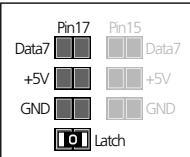
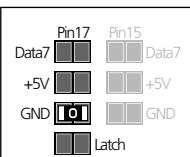
*Caution: make sure that *only one* position is closed in this solder jumper field. Other combinations are not allowed and cause damage to the board!

2.7.2 Solder Jumper Field B – Configuring Pin (17) of EXTENSION 2 Socket Connector

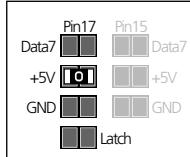
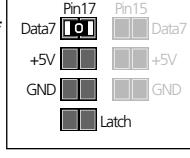
The solder jumper field B is located on the lower side of the RTC6 PCIe Board, see [Figure 6](#).

It serves to configure the signal at pin (17) of the EXTENSION 2 socket connector, see the following table.

See also ["Configuration by Solder Jumpers"](#), page 79.

Allowed jumper setting	Output at the EXTENSION 2 socket connector pin (17)
 open open open open	No signal.
 open open open closed*	LATCH signal.
 open open closed open	GROUND (low level).

**Caution: make sure that only one position is closed in this solder jumper field. Other combinations are not allowed and cause damage to the board!*

Allowed jumper setting (cont'd)	Output at the EXTENSION 2 socket connector pin (17) (cont'd)
 open closed open open	+5 V (high level).
 closed* open open open	DATA7 ^(a) .

**Caution: make sure that only one position is closed in this solder jumper field. Other combinations are not allowed and cause damage to the board!*

(a) Synonym: Data Bit #7. MSB of the 8-bit output value.

Notes

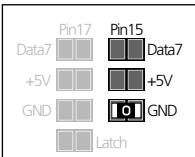
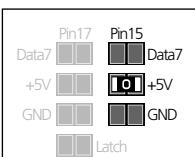
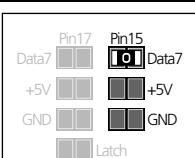
- Configurations of solder jumper field B and solder jumper field C are independent from each other.
- On RTC6 Ethernet Boards, the printed label of solder jumper field B is 'Pin17'. This has been chosen deliberately in order to keep consistency with already existing RTC boards. Even though with RTC6 Ethernet Boards pin (09) of the EXT. 2 socket connector, see [Chapter 16.2.12 "EXT. 2 Socket Connector"](#), page 873, is actually configured.

2.7.3 Solder Jumper Field C – Configuring Pin (15) of EXTENSION 2 Socket Connector

The solder jumper field C is located on the lower side of the RTC6 PCIe Board, see [Figure 6](#).

It serves to configure the signal at pin (15) of the EXTENSION 2 socket connector, see the following table.

See also "[Configuration by Solder Jumpers](#)", page 79.

Allowed jumper setting	Output at the EXTENSION 2 socket connector pin (15)
	open open open No signal.
	closed* open open GROUND (low level).
	open closed* open +5 V (high level).
	open open closed* DATA 7 ^(a) .

^{*} Caution: make sure that *only one* position is closed in this solder jumper field. Other combinations are not allowed and cause damage to the board!

(a) Synonym: Data Bit #7. [MSB](#) of the 8-bit output value.

Notes

- Configurations of solder jumper field C and solder jumper field B are independent from each other.
- On RTC6 Ethernet Boards, the printed label of solder jumper field B is 'Pin15'. This has been chosen deliberately in order to keep consistency with already existing RTC boards. Even though with RTC6 Ethernet Boards pin (08) of the EXT. 2 socket connector, see [Chapter 16.2.12 "EXT. 2 Socket Connector", page 873](#), is actually configured.



2.8 Accessories for the RTC6 PCIe Board

Only hardware extensions from SCANLAB should be used in combination with the RTC6 PCIe Board. In addition to the RTC6 PCIe Board and its software package, the following accessories can be obtained:

- [XY2-100 Converter, page 43](#)
- [Laser Adapter, page 43](#)
- [Data Cables, page 43](#)
- [Slot Cover with 9-pin D-SUB Connector for "2. SCANHEAD" Socket Connector, page 44](#)
- [Slot Cover with 15-pin D-SUB Connector for "MARKING ON THE FLY" Socket Connector, page 44](#)
- [Slot Cover with 15-pin D-SUB Connector and 9-pin D-SUB Connector, page 45](#)

Notes

- On the UFP Ext Board for the RTC6 PCIe Board⁽¹⁾, see [Chapter 17 "Appendix B: The UFP Ext Board", page 905](#).
- The following component can be used with RTC6 PCIe Boards:
 - "RTC5/6 varioSCAN 40 FLEX Extension" extension board (#128683)⁽²⁾
- The following components *cannot* be used with RTC6 PCIe Boards:
 - EXT1 ext board for the RTC5 (#123804)⁽³⁾
 - ADC add-on board (#121126)
 - "RTC4 STEP MOTOR EXTENSION" extension board (#112097)
 - I/O ext board for RTC3 and RTC4, (#108285) (#121721)

(1) Recommended for users with ANALOG controlled lasers, who want to achieve pixel output frequencies > 100 kHz, see [set_pixel_line](#).

(2) Can also be used with RTC6 Ethernet Boards. Requires Y cable #116050 (see [Figure 79](#)) and flat ribbon cable #105446. In addition, the customer has to supply a 1:1 cable (plug 1: identical in construction to Würth 61201023021, 10-pin, female; plug 2: see under Notes, [page 871](#)).

(3) Developed for RTC5. Cannot be used with RTC6 PCIe Boards due to mechanical reasons (collision with cooling element; replacement: UFP Ext Board, [page 905](#)).

2.8.1 XY2-100 Converter

The [XY2-100 Converter \(Accessory\)](#) allows the RTC6 PCIe Board to control scan systems which are equipped with a conventional XY2-100 interface.

2.8.2 Laser Adapter

The SCANLAB laser adapter is plugged into the 15-pin [LASER Connector](#) of the RTC6 PCIe Board. Then a 9-pin female D-SUB connector of this adapter provides the same signals and pin-out as the 9-pin LASER connector of the RTC4/RTC5. See also [Section "Laser Adapter \(Accessory\)", page 75](#).

2.8.3 Data Cables

To connect the RTC6 PCIe 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 also [Chapter 4.5.3 "Data Cables \(Accessories\)", page 70](#).

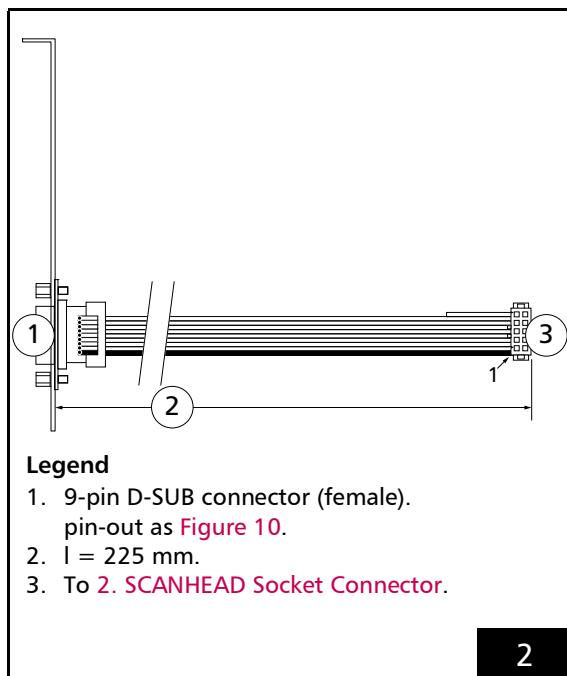
2.8.4 Slot Cover with 9-pin D-SUB Connector for "2. SCANHEAD" Socket Connector

For connecting a second scan head or a z axis to the [2. SCANHEAD Socket Connector](#) a slot cover is available, see [Figure 2](#):

- #115132

Its 9-pin D-SUB connector has the same pin-out as the [SCANHEAD Connector](#), see [Figure 10](#).

See also [Section "Second Scan Head Slot Cover \(Accessory\)"](#), page 67.



Second Scan Head Slot Cover (Accessory) #115132.

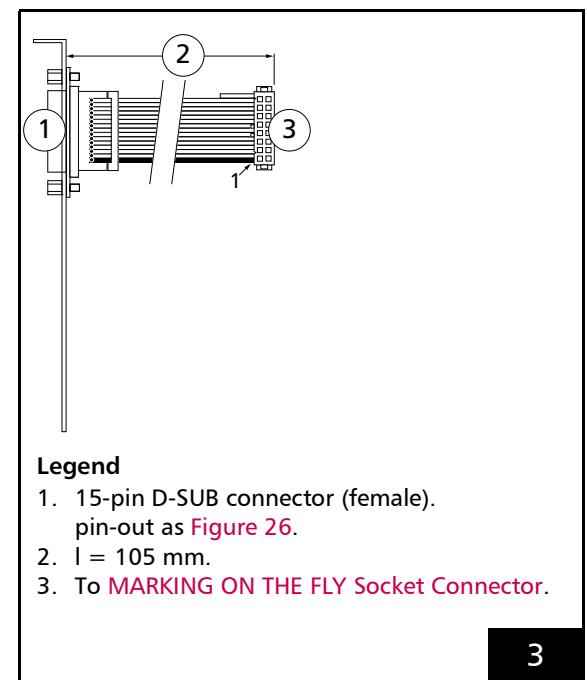
2.8.5 Slot Cover with 15-pin D-SUB Connector for "MARKING ON THE FLY" Socket Connector

For using the input ports and signals of the [MARKING ON THE FLY Socket Connector](#), a slot cover is available, see [Figure 3](#):

- 109272

The pin-out of its 15-pin D-SUB connector (female) is shown in [Figure 26](#).

See also [Section "MARKING ON THE FLY Slot Cover \(Accessory\)"](#), page 82.

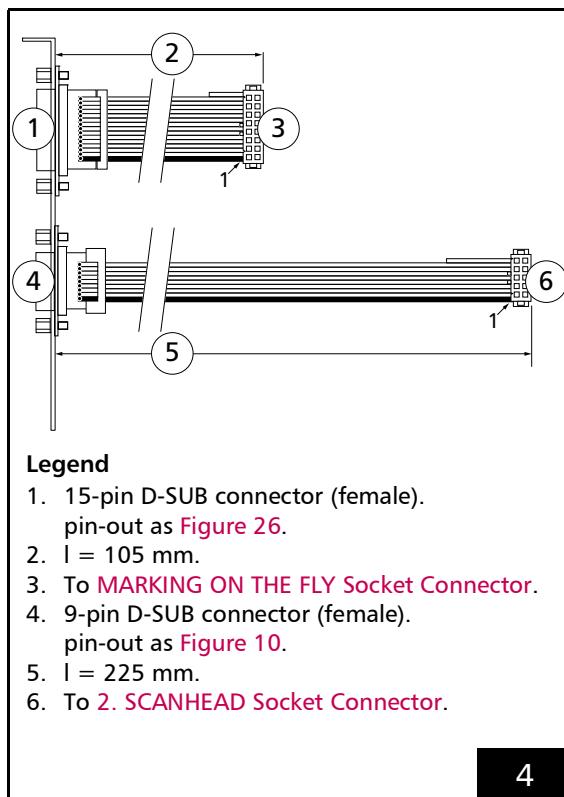


MARKING ON THE FLY Slot Cover (Accessory) 109272.

2.8.6 Slot Cover with 15-pin D-SUB Connector and 9-pin D-SUB Connector

The connectors of #115132 and 109272 together on a single bracket (for the same purpose as these) features bracket, see [Figure 4](#):

- #130209



Slot cover #130209.

2.8.7 Extension Board "RTC5/6 varioSCAN FLEX Extension"

For RTC5/6 Boards, the "RTC5/6 varioSCAN 40 FLEX Extension" extension board (#128683) is available⁽¹⁾.

It has been specially developed to control analog and digital varioSCAN 40 FLEX devices by 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 RTC5/6 varioSCAN FLEX Extension for RTC5 and RTC6 Control Boards".

Notes

- "RTC5/6 varioSCAN 40 FLEX Extension" extension board (#128683) does *not* support [move_to](#)⁽²⁾.

(1) In contrast to "RTC4 STEP MOTOR EXTENSION" extension board (#112097) – which can also be used with RTC5 Boards – it has the advantage that the EXTENSION 1 socket connector remains unoccupied.

(2) [move_to](#) has been introduced for "RTC4 STEP MOTOR EXTENSION" extension board (#112097).



2.9 Supplementary Software

To facilitate customizing RTC correction files basing on data of your own test measurements, SCANLAB offers the correXion pro software with accompanying manual, see also [Section "Image Field Correction Algorithm", page 173](#).

By using the SCANLAB laserDESK software, own laser marking and material-processing programs ("laser jobs") can be created and executed without software development. Many of the RTC6 functions are supported.⁽¹⁾

For further information on laserDESK, refer to the SCANLAB homepage.

(1) See also [Notice!](#), page 72.

2.10 Notes for RTC4 Users

This chapter provides an overview of key changes introduced with RTC6 PCIe Boards for comparable functions of RTC4 boards.

For example, [Chapter 2.10.2 "Porting RTC4 Source Code to the RTC6 PCIe Board", page 48](#) discusses a possible approach to porting RTC4 programs to run on the RTC6 PCIe Board.

The individual command descriptions in particular note changes.

2.10.1 Hardware Changes

When migrating from the RTC4 to the RTC6 PCIe Board, you need to consider the following hardware changes for correct cabling of the system components.

Controlling Scan Systems

- To control scan systems with XY2-100 interface or XY2-100 Enhanced interface, you additionally need the [XY2-100 Converter \(Accessory\)](#).
- The RTC6 PCIe Board *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 [Chapter 4.5.3 "Data Cables \(Accessories\)", page 70](#).
- To use the second scan head connector, you need a different adapter cable (including slot cover available from SCANLAB), see [Section "Second Scan Head Slot Cover \(Accessory\)", page 67](#).
- For controlling one 3-axis scan system, both scan head connectors must be used, see [Chapter 8.5.2 "3D Scan Systems", page 236](#).
- Simultaneous control of two 3-axis scan systems requires two RTC6 PCIe Boards, see [Chapter 8.5.2 "3D Scan Systems", page 236](#).

Controlling the Laser

- The [LASER Connector](#) at the RTC6 PCIe Board slot cover has 15 pins (RTC4: 9 pins). Its pin-out is not jumper-configurable.
 - The RTC6 PCIe Board does not require jumpers **X6** and **X7** of the RTC4 because all signals are available at the RTC6 PCIe Board [LASER Connector](#).
 - The voltage range of the analog outputs is always 0...10 V (0 V...2.50 V is no longer supported; the jumper **X3** of the RTC4 does not exist on the RTC6 PCIe Board).
- If you want to connect a laser to the RTC6 PCIe Board 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 SCANLAB laser adapter, see [Section "Laser Adapter \(Accessory\)", page 75](#), 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 defined by an RTC6 command, see [set_laser_control](#).
 - Jumper **X10** of the RTC4 does not exist on the RTC6 PCIe Board.

EXTENSION 1 Socket Connector

- The RTC6 PCIe Board EXTENSION 1 socket connector is – except for the additionally provided signals at pin (33)...(35) – identical to the EXTENSION 1 socket connector of the RTC4, see [Figure 22](#).
- With the RTC6 PCIe Board, 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 [Section "Configuring the Output Signal Level", page 77](#).

EXTENSION 2 Socket Connector

- The RTC4 EXTENSION 2 socket connector does not exist on the RTC6 PCIe Board. Therefore, the "I/O extension board for RTC3 and RTC4" (#108285, #121721) cannot be attached to the RTC6 PCIe Board.
- The RTC6 PCIe Board EXTENSION 2 socket connector is – except for the LATCH signal at pin (17) which can be optionally set – identical to the LASER EXTENSION socket connector of the RTC4, see [Figure 24](#).
 - For configuring pin-outs the RTC6 PCIe Board provides the solder jumper field B and C, see also [Section "Configuration by Solder Jumpers", page 79](#). These correspond to RTC4 jumpers X8 and X9.

MARKING ON THE FLY Socket Connector

- The RTC6 MARKING ON THE FLY socket connector and the RTC4 MARKING ON THE FLY socket connector are identical, see [Figure 25](#). Observe the safety notice on encoder counter direction, [page 56](#).

Other Hardware Interfaces

- The following interfaces only exist on the RTC6 PCIe Board:
 - Master and Slave, see [Chapter 4.4 "Master Socket Connector, Slave Socket Connector", page 65](#)
 - RS232, see [Chapter 4.6.5 "RS232 Socket Connector", page 82](#)
 - McBSP/ANALOG, see [Chapter 4.6.6 "McBSP/ANALOG Socket Connector", page 83](#)
 - STEPPER MOTOR, see [Chapter 4.6.7 "STEPPER MOTOR Socket Connector", page 86](#)

2.10.2 Porting RTC4 Source Code to the RTC6 PCIe Board

User programs written for the RTC4 can only run on the RTC6 PCIe Boards after suitable code revision. This applies even when the actual program flow remains unchanged.

Changed Initialization

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

- At the beginning of the program, an `init_rtcl_dll` must be inserted for initializing the [RTC6 DLL](#) and RTC6 board management, see [Chapter 6.2.3 "Initializing the RTC6 DLL and RTC6 Board Management", page 96](#).
- The files for initializing the board by `load_program_file` are different 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 [Chapter 7.3.5 "Image field Correction and Correction Tables", page 172](#).
- For laser control initialization, `set_laser_control` must be additionally inserted, see [Chapter 7.4 "Laser Control", page 183](#).

Command Changes

All unsupported RTC4 commands must be removed or replaced. See also [Chapter 10.3 "Unsupported RTC4/RTC5 Commands"](#), page 846.

Changed or enhanced RTC4 commands must be handled differently in the program (for example, by modifying supplied parameter values or evaluating returned values differently). Changes to still supported commands are listed in the individual command descriptions (in [Chapter 10.2 "RTC6 Command Set"](#), page 314), "RTC4→RTC6" row.

RTC4 commands that need to be replaced or checked are:

- `auto_cal` changed
- `auto_change_pos` changed
- `control_command` changed
- `dsp_start` not supported
- `get_head_status` changed
- `get_hi_data` changed
- `get_list_space` changed
- `get_marking_info` changed
- `get_RTC_version` changed
- `get_startstop_info` changed
- `get_status` changed
- `get_value` changed
- `get_waveform` changed
- `get_xy_pos` not supported
- `get_xyz_pos` not supported
- `goto_xy` changed
- `goto_xyz` changed
- `list_jump_cond` changed
- `list_nop` changed
- `load_correction_file` changed
- `load_program_file` changed
- `read_pixel_ad` not supported
- `read_status` changed
- `rtc4_count_cards` not supported
- `select_cor_table` changed
- `select_list` not supported
- `select_RTC` changed
- `set_control_mode` enhanced
- `set_control_mode_list` enhanced
- `set_laser_mode` enhanced
- `set_laser_timing` changed
- `set_list_mode` not supported
- `set_matrix` changed
- `set_matrix_list` changed
- `set_offset` changed
- `set_offset_list` changed
- `set_piso_control` not supported
- `set_pixel` changed
- `set_pixel_line` changed
- `set_softstart_level` changed
- `set_softstart_mode` changed
- `set_trigger` enhanced
- `set_wobbel` changed
- `set_wobbel_xy` 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 RTC6 PCIe Board – 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 RTC6 PCIe Board (whereas with 16-bit resolution for the RTC4), see also [Chapter 7.3.2 "Image Field Size and Image Field Calibration", page 166](#).
- Moreover, for Processing-on-the-fly applications, an extended, 29-bit value range is available with the RTC6 PCIe Board, see also [Chapter 7.3.3 "Virtual Image Field", page 168](#).
- For commands such as `write_da_x`, analog output values are specified with 12-bit resolution for the RTC6 PCIe Board (whereas with 10-bit resolution for the RTC4).
- For `set_laser_timing`, output period and pulse length are specified with $1/64 \mu\text{s}$ resolution for the RTC6 PCIe Board (whereas with $1/8 \mu\text{s}$ or $1 \mu\text{s}$ resolution for the RTC4).
- For `set_laser_delays`, the **Laser Delays** are specified with $1/64 \mu\text{s}$ resolution for the RTC6 PCIe Board (whereas with $1 \mu\text{s}$ resolution for the RTC4).

Alternatively, the **RTC6 DLL** can be set to the **RTC4 Compatibility Mode** by `set_rtc4_mode`. Then the **RTC6 DLL** automatically converts parameter values so that many RTC4 command sequences can run unchanged on the RTC6 PCIe Board.

RTC4 Compatibility Mode affects the following RTC4 commands (descriptions of the respective commands include relevant information under the heading "RTC4→RTC6"):

- `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`

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 be executed in **RTC6 Standard Mode** or **RTC4 Compatibility Mode**.



Changed Timing Behavior

The following RTC4 list commands are processed by the RTC6 PCIe Board as "short list commands". This can result in a changed timing behavior during user program execution, see [Section "Normal, Short, Variable and Multiple List Commands", page 301](#).

- `clear_io_cond_list`
- `list_call`
- `list_call_cond`
- `list_jump_cond` (changed)
- `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` (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 [Section "Automatic Delay Adjustments", page 154](#).

2.10.3 New and Changed Functionality

Interface to the PC

- The RTC6 board driver supports simultaneous control of any number of RTC6 PCIe Boards in a single PC, see [Chapter 6.6 "Using Several RTC6 PCIe Boards in One PC"](#), page 122.
- Connectors and commands for master/slave synchronization of several RTC6 PCIe Boards, see [Chapter 4.4 "Master Socket Connector, Slave Socket Connector"](#), page 65.

Controlling Scan Systems

- New interface to the scan system, see [Chapter 4.5.1 "Scan Head Connectors and Transfer Protocol"](#), page 66:
 - 9-pin female D-SUB connector at the RTC6 PCIe Board slot cover and 10-pin socket connector
 - SL2-100 protocol
 - 2 data channels each for both scan head connectors
 - Galvanically isolated signals
 - 20-bit positioning resolution, see [Chapter 7.3.2 "Image Field Size and Image Field Calibration"](#), page 166
 - Enhanced status return from the scan system, see [Chapter 7.3.7 "Status Monitoring and Diagnostics"](#), page 182
 - Control channels and status channels for enhanced data transfer with *iDRIVE* scan systems⁽¹⁾
- An [XY2-100 Converter \(Accessory\)](#) is available for data transfer according to the XY2-100 protocol.
 - 25-pin female D-SUB connector
 - 16-bit positioning resolution
 - Status return according XY2-100 protocol or XY2-100 Enhanced protocol
 - Transfer synchronization is configurable for long data cables by solder jumpers in the [XY2-100 Converter \(Accessory\)](#)

The RTC6 PCIe Board provides power for the [XY2-100 Converter \(Accessory\)](#).

- For optical data transfer between the RTC6 PCIe Board and scan systems, no special variant of the RTC6 PCIe Board (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 [Chapter 4.5.3 "Data Cables \(Accessories\)"](#), page 70.
- For controlling a 3-axis scan system, see [Chapter 8.5.2 "3D Scan Systems"](#), page 236:
 - Both scan head connectors must be used
 - Simultaneous control of two 3-axis scan systems requires 2 RTC5 PCI Boards
- [Image field correction](#)
 - New correction files are needed, see [Chapter 7.3.5 "Image field Correction and Correction Tables"](#), page 172:
 - File name extension ".ct5"
 - Correction with higher resolution
 - Queryable information in the correction file header
 - For the RTC6 PCIe Board, RTC4 correction files (.ctb) need to be newly calculated or converted by [CorrectionFileConverter.exe](#) which is part of the RTC6 Software Package.
 - Up to 8 correction files can be loaded to the RTC6 PCIe Board
 - Enhanced 3D image field correction by stretch correction tables, see [Section "Enhanced 3D Correction"](#), page 239
- Coordinate transformations, see [Chapter 8.2 "Coordinate Transformations"](#), page 223:
 - The correction file is no longer transformed (rotation, shift, extension) at download
 - Matrix transformations are only applied after [Microstepping](#) – this may cause the mark speed to change
 - ["Local Online Positioning"](#), see [Chapter 8.3.1 "Local Online Positioning"](#), page 227

(1) See Glossary entry on [page 26](#).

- Coordinate transformations in the virtual **Image field** (incl. "Global Online Positioning"), see **Chapter 7.3.3 "Virtual Image Field"**, page 168
 - 29-bit position coordinates (virtual **Image field**): objects larger than the real **Image field** are possible
- Position monitoring of **iDRIVE** scan systems⁽¹⁾ by backward transformation of actual position values, see **Chapter 8.1.3 "Monitoring the Positioning"**, page 213
- Automatic self-calibration, see **Chapter 8.10 "Automatic Self-Calibration"**, page 274:
 - Optimization of previous functions
 - ASC hardware check
- **Jump Mode**, see **Chapter 8.1.5 "Jump Mode"**, page 216
- Cycle synchronization, see **Chapter 7.4.10 "Synchronization of the RTC6 Clock Cycle and an External Clock Signal"**, page 206

Controlling the Laser

- The signal levels of the laser control signals are no longer determined by a jumper configuration. Instead, they are software-configured, see **set_laser_control**
- **LASER Connector** with all laser control signals at the RTC6 PCIe Board slot cover, see **Chapter 4.6.1 "LASER Connector"**, page 72, 9-pin female D-SUB connector only by the SCANLAB laser adapter, see **Section "Laser Adapter (Accessory)"**, page 75
- **LASER Connector** configurable by software command, see **Chapter 7.4.2 "Configuring the LASER Connector"**, page 186
 - Laser control signals with 15 ns resolution and 20 mA output current
- Standby signals in YAG modes, see **Chapter 7.4.4 "YAG Modes 1, 2, 3, 5"**, page 189
- YAG Mode 5: Time between FirstPulseKiller signal and first laser pulse in YAG mode is freely programmable, see **Chapter 7.4.4 "YAG Modes 1, 2, 3, 5"**, page 189
- **Laser Mode 6**: LASERON signal synchronized with a continuously-running LASER1 signal, see **Chapter 7.4.6 "Laser Mode 6"**, page 193

- **Pulse Picking Laser Mode**, see **Chapter 7.4.8 "Pulse Picking Laser Mode"**, page 195
- Laser pulse period, pulse length or analog output are also programmable within a **Polyline** between two vectors – where the laser remains on⁽²⁾, see "short list commands" in **Section "Normal, Short, Variable and Multiple List Commands"**, page 301
- Commands for position-dependent, speed-dependent, vector-defined and encoder-speed-dependent laser control, see **Chapter 7.4.9 ""Automatic Laser Control""**, page 196

Interfaces for Peripheral Equipment

- 16-bit digital output, see **Section "16-Bit Digital Input Port and 16-Bit Digital Output Port"**, page 77, and **Chapter 9.1.1 "16-Bit Digital Output Port"**, page 281:
 - 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 port, see **Section "8-Bit Digital Output Port"**, page 80 and **Chapter 9.1.2 "8-Bit Digital Output Port"**, page 282:
 - 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 output ports, see **Section "12-Bit Analog Output Port 1 and 2"**, page 73, and **Chapter 9.1.4 "12-Bit Analog Output Port 1 and 2"**, page 282:
 - 12 bit resolution
 - 0...10 V (0 V...2.50 V no longer available)
 - Adjustable "stop output value"
- 16-bit digital input port, see **Section "16-Bit Digital Input Port and 16-Bit Digital Output Port"**, page 77, and **Chapter 9.2.1 "16-Bit Digital Input Port"**, page 287.
 - SYNC signal for synchronization of data transmission

(1) See Glossary entry on [page 26](#).

(2) Is switched off with the RTC4.

- Programmable debouncing of external start signals, see [bounce_supp](#) and [Section "External Start", page 289](#)
- Regular (periodic) [External Starts](#), see [Section "Regular \(Periodic\) External Starts", page 292](#)
- New interfaces:
 - 2-bit digital input port and 2-bit digital output port at the [LASER Connector](#), see [Section "2-Bit Digital Input Port", page 73](#) and [Section "2-Bit Digital Output Port", page 73](#)
 - RS-232 interface by an 10-pin socket connector, see [Chapter 4.6.5 "RS232 Socket Connector", page 82](#)
 - Stepper motor signals for 2 motors by an on-board 10-pin socket connector, see [Chapter 4.6.7 "STEPPER MOTOR Socket Connector", page 86](#)
 - McBSP and ANALOG IN by a 10-pin socket connector, see [Chapter 4.6.6 "McBSP/ANALOG Socket Connector", page 83](#)
- For the RTC6 PCIe Board, there is no longer an 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 RTC6 PCIe Board EXTENSION 2 socket connector corresponds to the RTC4 "LASER EXTENSION" socket connector).

General Programming

- Utility files for C, C++, C# and Delphi, see [Chapter 6.2.2 "Importing Commands", page 94](#), but no longer utility files for Basic
- Commands for changing access rights to RTC6 PCIe Boards, see [Chapter 6.7 "Usage of RTC6 PCI Express Boards by Several User Programs", page 127](#)
- Improved and extended list handling, see [Chapter 6.4 "List Handling", page 104](#)
 - List memory with 8,388,608 storage positions
 - List memory free configurable (in two list memory areas and a protected list memory area)
 - Defining protected subroutines
 - Loading lists with protection
 - Loops in lists and subroutines
 - RTC4-Circular queue mode is not available (but can be coded alternatively)
 - List Status
 - List Execution Status
- "Short" list commands (for example, 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 cycle, see [Section "Normal, Short, Variable and Multiple List Commands", page 301](#))
- Functions for error handling and download verification, see [Chapter 6.8 "Error Handling", page 129](#)

Laser Marking

- Vectors and arcs
 - Commands for marking ellipses, see [Section "Ellipse Commands", page 137](#)
 - Commands for marking helices, see [Chapter "3D Commands", page 237](#)
 - Timed arc commands, timed 3D vector commands, see [Chapter 8.9 "Timed Commands", page 273](#)
 - Para-Mark commands and Para-Jump commands for "vector-controlled laser control", see [Section "Vector-Defined Laser Control", page 205](#)
- Characters and texts
 - Defining character sets and text strings, see [Section "Defining Indexed Character Sets", page 117](#)
 - Commands for marking individual characters and for marking texts (with selectable character set), see [Section "Calling Indexed Characters", page 118](#)
 - Commands for marking dates, times and serial numbers (with selectable character set and selectable serial-number-set), see [Chapter 7.5 "Marking Dates, Times and Serial Numbers", page 209](#)
- `micro_vector[*]` commands (direct position output without Microstepping), see [Chapter 8.8 "micro_vector\[*\] Commands", page 272](#).

Special Functions

- Synchronization of scan system and laser control
 - Sky Writing, see [Chapter 7.2.4 "Sky Writing", page 159](#)
- Pixel Output Mode (marking of bitmaps), see [Chapter 8.7 "Pixel Output Mode", page 262](#):
 - Pixel output frequencies up to 800 kHz, beyond that with [Option "UFPM"](#) up to 3.2 MHz
 - RTC4-Pixel mode 0 with 10 μ s clock synchronous output is no longer supported
 - 15 ns resolution
 - 0...100% laser pulse length
 - Pixel-Mode 0 not supported
 - Reading of analog voltages is not supported
 - 3D pixel lines with [set_pixel_line_3d](#)
- Commands for conditional execution of any list command, see [Chapter 9.3.2 "Conditional Command Execution", page 294](#)
- 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. With ["Freely Definable Wobble Shapes"](#), also the laser power can be varied, see [Chapter 8.4 "Wobble Mode", page 231](#)
- Camming functionality, see [Chapter 8.11 "Camming", page 278](#)
- Enhanced signal recording, see [set_trigger](#) and [set_trigger4](#)

- Processing-on-the-fly, see [Chapter 8.6 "Processing-on-the-fly", page 241](#)
 - two encoder input ports (RS-422) with 32-bit counter for Processing-on-the-fly correction with encoder signals on 2 axes, see [Chapter 9.3.3 "Synchronization by Encoder Signals", page 297](#); alternatively: Processing-on-the-fly correction with McBSP signals, see [Chapter 9.3.4 "Synchronization and Online Positioning by McBSP Signals", page 299](#)
 - 29-bit coordinates (virtual **Image field**): objects larger than the real **Image field** are possible, see [Chapter 7.3.3 "Virtual Image Field", page 168](#), and [Chapter 8.6.6 "Virtual Image Field with Processing-on-the-fly", page 251](#)
 - Up to 8 objects within the Processing-on-the-fly track delay (between trigger and marking position), see [Section "External Start", page 289](#)
 - Accurate "External 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-positioning stage)
 - Encoder-based Processing-on-the-fly correction for the z axis ("FlyZ correction"), see [Chapter 8.6.11 "Processing-on-the-fly Correction for the z Axis", page 256](#)

Notice!

- The encoder counter direction of RTC4 boards (includes RTC4 Ethernet Board and RTC4 SCANalone Board) is opposite to that of RTC5 boards and RTC6 boards. Therefore, when changing RTC-control board you need to either adapt the cabling or your user program accordingly.

2.11 Notes for RTC5 Users

RTC6 PCIe Board can control:

- Scan systems with SL2-100 interface
- Scan systems with XY2-100 interface
(in conjunction with the **XY2-100 Converter (Accessory)**)

The following is an overview of key differences between RTC6 PCIe Boards and RTC5 boards.

Also described are possible approaches for migrating RTC5 program code for use with RTC6 PCIe Boards.

Notes

- Users of excelliSCAN *scan heads* must *additionally* read the "**excelliSCAN Scan Heads – Functional Principle of SCANAhead Servo Control and Operation by RTC6 Boards**" Manual.

2.11.1 RTC6 PCIe Boards vs. RTC5 Boards

Compared to the RTC5 board family, the RTC6 PCIe Board uses newer and more powerful hardware components. This applies in particular to the **DSP**, the **FPGA** and the main memory.

Existing user programs only require a few changes, see [Chapter 2.12.2 "Adapting RTC5 Source Code for the RTC6 PCIe Board"](#), page 59.

Functionality changes

- The RTC6 PCIe Board executes short vectors⁽¹⁾ faster than RTC5 boards, see [set_dsp_mode](#).
- The RTC6 PCIe Board list memory features 8,388,608 (= 2^{23}) list positions. List 1 and List 2 are each initialized with 4,194,304 (= 2^{22}) list positions.
- The **Pixel Output Mode** is significantly enhanced, see [set_pixel_line](#).
 - RTC6 PCIe Boards without **Option "UFPM"** can achieve pixel output frequencies of up to 800 kHz out-of-the-box.
 - RTC6 PCIe Boards, with **Option "UFPM"**, even up to 3.2 MHz.
 - With [set_pixel_line](#) it is now possible to digitally output laser power control values. This can be done either at the 8-bit or 16-bit output (EXTENSION 2 socket connector or EXTENSION 1 socket connector).
- Extended measurement-value recording with the RTC6 PCIe Board: 4 recording channels, each storing 8,388,608 data values, see [set_trigger4](#). Practically any amount of data can be recorded (ring buffer).

(1) "dashed lines"

- 8 3D correction tables can be used on the RTC6 PCIe Board simultaneously with 4 recording channels (see `set_trigger`, `set_trigger4`) and the complete list memory. With `number_of_correction_tables`, a user-defined limit to less than 8 can be set to check for incorrect user input.
- The virtual `Image` field of the RTC6 PCIe Board is now ± 28 bit = 29 bit.
- You can set an appropriate `Tracking error` compensation value for 3D systems with different `Tracking error` behavior between the xy axes and z axis, see `set_timelag_compensation`.
- The output synchronization of the RTC5 is replaced by a cycle synchronization with the RTC6 PCIe Board, see [Chapter 7.4.10 "Synchronization of the RTC6 Clock Cycle and an External Clock Signal", page 206](#).

Parameter changes

- `Laser Delays` (parameter `LaserOnDelay` and `LaserOffDelay` of `set_laser_delays`) as well as `LaserOnShift` with `Sky Writing` commands have a higher resolution of 1/64 μ s.
For downward compatibility with RTC5 boards, see [step 3, page 60](#).
- In fact the parameter `Timelag` of `set_sky_writing_para` is specified in 64-bit IEEE floating point format ("double"; 1.0 = 1 μ s) but on the RTC6 PCIe Board it is actually executed with a resolution of 1/64 μ s resolution.

2.11.2 RTC6 PCIe Boards and Scan Systems with SCANAhead Servo Control

- Only the RTC6 PCIe Board allows usage of SCANAhead servo control, for example, offered by `scan heads` of the excelliSCAN series.
- If you configure the RTC6 PCIe Board by `set_scanahead_params` for operation with an excelliSCAN scan head, the laser control then automatically takes the SCANAhead servo technology's `PreviewTime` time into account (that is, processing time for calculating and executing scanner trajectories⁽¹⁾). This ensures fuller usage of dynamics and precision.
- `activate_scanahead_autodelays` lets you simply and quickly place the excelliSCAN into operation. Neither `Laser Delays` nor `Scanner Delays` need to be determined or set.
- For further details, see the ["excelliSCAN Scan Heads – Functional Principle of SCANAhead Servo Control and Operation by RTC6 Boards" Manual](#), which also describes SCANAhead-specific commands.

(1) See Glossary entry on [page 29](#).

2.11.3 Restrictions with RTC6 PCIe Boards

Availability of Technical Variants

RTC6 boards are available as RTC6 PCIe Boards and RTC6 Ethernet Boards.

Peripheral Interfaces

The ADC add-on card (#121126, optional accessory for the RTC5 PCI board) cannot be used with the RTC6 PCIe Board. The RTC6 PCIe Board supplies the two 10 V analog input ports **ANALOG IN0** and **ANALOG IN1** at the McBSP/ANALOG socket connector directly, see [Chapter 4.6.6 "McBSP/ANALOG Socket Connector", page 83](#).

Functionality

With RTC6 PCIe Boards, the McBSP/ANALOG socket connector cannot be used in **SPI** mode, see also [Chapter 4.6.6 "McBSP/ANALOG Socket Connector", page 83](#).

Known restrictions are mentioned in [RTC6_Release_Notes_YYYY_MM_DD.pdf](#).

2.12 Source Code for RTC6 User Programs

2.12.1 Creating New RTC6 Source Code

To create new source code for RTC6 user programs, proceed as described in [Chapter 6.2 "Initialization and Program Start-Up", page 94](#) while observing [Chapter 2.12.2 "Adapting RTC5 Source Code for the RTC6 PCIe Board", page 59](#).

2.12.2 Adapting RTC5 Source Code for the RTC6 PCIe Board

User programs written for the RTC5 only become usable on the RTC6 PCIe Board, if you modify the program code as described in the following.

This is true even if program flow remains unchanged.

Mandatory Steps

(1) Ensure the following prerequisites are met:

- RTC6 PCIe Board is installed
 - RTC6 board driver is installed for your operating system
 - The following files are present:
 - RTC6DLL.dll** or **RTC6DLLx64.dll**,
 - RTC6OUT.out** (with RTC6 PCIe Boards) or **RTC6ETH.out** (with RTC6 Ethernet Boards),
 - RTC6RBF.rbf**,
 - RTC6DAT.dat**,
 - * **.CT5** (correction file(s))
- Note:** **RTC6OUT.out**/**RTC6ETH.out**, **RTC6RBF.rbf**, **RTC6DAT.dat** must all be in the same folder.

- The import declarations were imported in your software and the calling convention `std_call` has been applied, see [Chapter 6.2 "Initialization and Program Start-Up", page 94.](#)
- (2) Take into account these command name changes of the RTC6 command set:
- Do *not* generally change `_rtc5_` to `_rtc6_`!
 - Change:
 - init_rtc5_dll to init_rtc6_dll
 - free_rtc5_dll to free_rtc6_dll
 - rtc5_count_cards to rtc6_count_cards
 - Notice: Do *not* generally change `set_rtc5_mode` to `set_rtc6_mode` – here see step 3!
- (3) `set_rtc6_mode` is set automatically at program start. Then an expanded parameter resolution is available for
- z coordinates of 20 bits
 - [Laser Delays](#) of 1/64 μ s
- In contrast, the control command `set_rtc5_mode` ensures downward compatibility of the parameters ("RTC5 Compatibility Mode").
- [Laser Delays](#):
In [RTC5 Compatibility Mode](#), the RTC6 board multiplies the values specified for `LaserOnDelay` and `LaserOffDelay` (of `set_laser_delays`) automatically by 32. This also applies to the values specified for `LaserOnShift` (of `set_sky_writing`) and `LaserOnShift` (of `set_sky_writing_para`).
 - z coordinates:
In [RTC5 Compatibility Mode](#), the RTC6 board multiplies the z coordinates and defocus values (for example, `Shift` of `set_defocus`) automatically by 16.

For further modifications and optimizations of your code, see the following chapter.

Optional Steps and Notes on Further Modification and Optimization

- Commands which have been removed from the RTC6 command set or are no longer supported must be removed or replaced in the program code, see [Chapter 2.13.2 "Removed from RTC6 Command Set", page 61.](#)
- Check if command changes might affect your user program, see [Chapter 2.13.1 "Changes to the RTC6 Command Set", page 61](#), and modify your program code accordingly.
- Take into account the changed behavior of `load_program_file`.
Notice: `load_program_file` now stops lists without warning. The single-board command can even be used again after a version conflict.
- **Note on enhanced parameter resolution**
Take into account the enhanced parameter resolution of z coordinates when using `set_rtc6_mode`. See also step 3, page 60.
- **Note on changed time behavior**
The replacement of the automatic delay adjustment can result in faster time behavior for short vectors.
By `set_dsp_mode(2)`, the original RTC5 time behavior can be restored, see [set_dsp_mode](#).

2.13 Changes to the RTC6 Command Set

The most RTC5 commands are covered in the RTC6 command set. The following chapter describes the few differences:

- *Changed* commands, see [Chapter 2.13.1 "Changes to the RTC6 Command Set", page 61](#)
- *Removed* commands, see [Chapter 2.13.2 "Removed from RTC6 Command Set", page 61](#)

list commands list command

2.13.1 Changes to the RTC6 Command Set

Meanings:

con	control command
nor	normal list command
ds	delayed short list command

(n_) config_list	con	343	
	get_dll_version	con	394
(n_) get_hex_version	con	404	
(n_) get_RTC_version	con	417	
(n_) get_status	con	422	
(n_) get_sync_status	con	426	
(n_) *init_fly_2d	con	454	
(n_) load_correction_file	con	485	
(n_) *load_fly_2d_table	con	492	
(n_) load_program_file	con	500	
(n_) *load_stretch_table	con	503	
(n_) mcbsp_init_spi	con	536	
(n_) number_of_correction_tables	con	542	
(n_) select_cor_table	con	594	
(n_) *set_auto_laser_control	con	604	
(n_) *set_auto_laser_params	con	608	
(n_) *set_auto_laser_params_list	ds	608	
(n_) set_dsp_mode	con	621	
(n_) *set_pixel_line	nor	703	
(n_) set_trigger4	nor	744	

* Note: RTC6 command with extended functionality.

2.13.2 Removed from RTC6 Command Set

free_RTC5_dll	846
init_RTC5_dll	846
RTC5_count_cards	846

3 Safety During Installation and Operation

Read these operating instructions completely before you proceed with installing and operating the RTC6 PCIe Board.

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

The following conventions apply to safety notices in this manual:



- Safety notices which draw attention to severely injuries or even death are identified by the hazard symbol and the signal word "Warning!".
- Safety notices which draw attention to a health hazard are identified by the hazard symbol and the signal word "Caution!".
- Safety notices that recommend proper use of the device or warn against possible damage to property are (without hazard sign) only identified by the signal word "Notice!".



Notice!

- For storage and operation of the board, avoid electromagnetic fields and static electricity. These can damage the electronics on the 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.

3.2 Laser Safety

The RTC6 PCIe Board 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.

3.1 Safe Operation

Notice!

- 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.

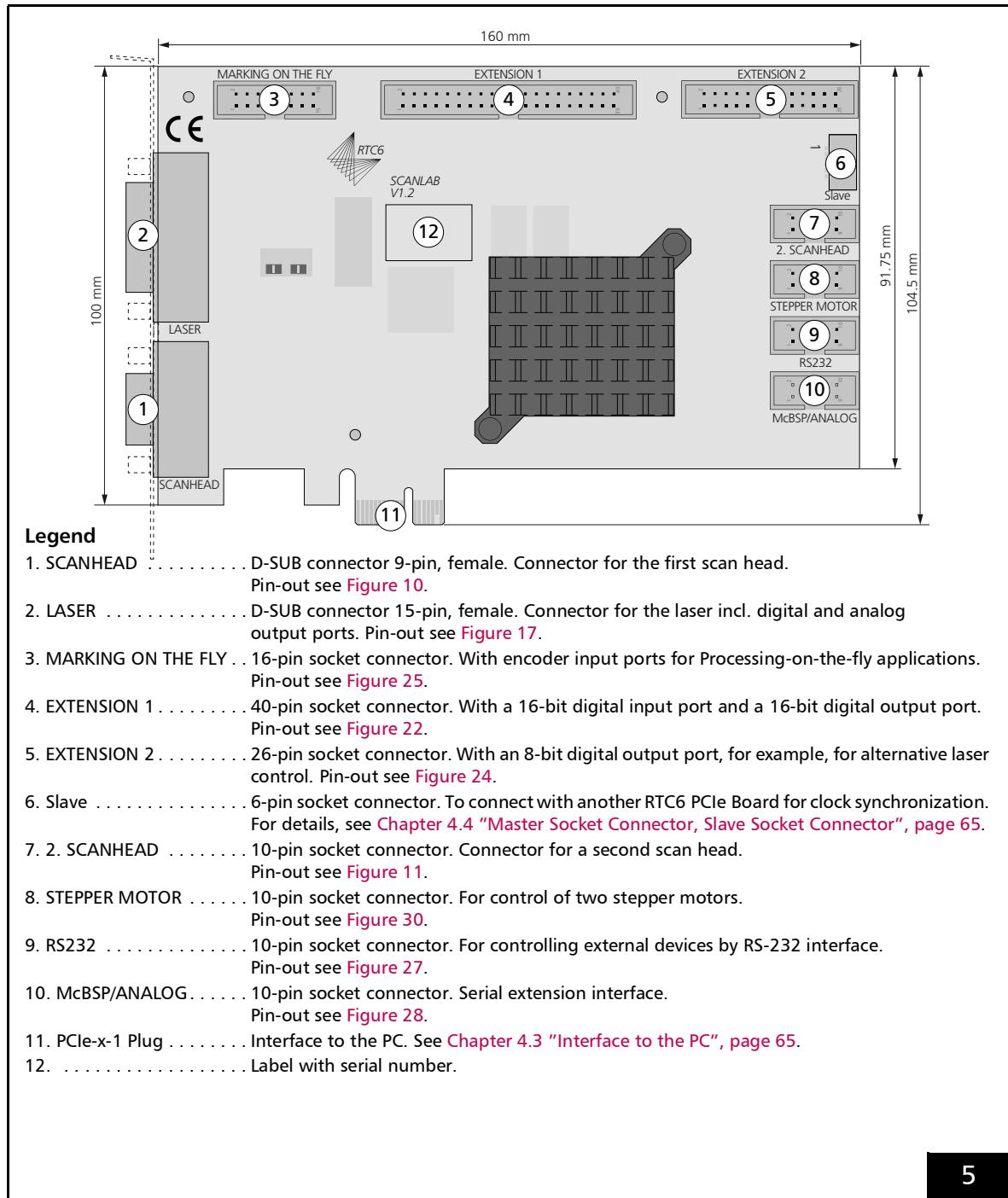


Caution!

- 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.

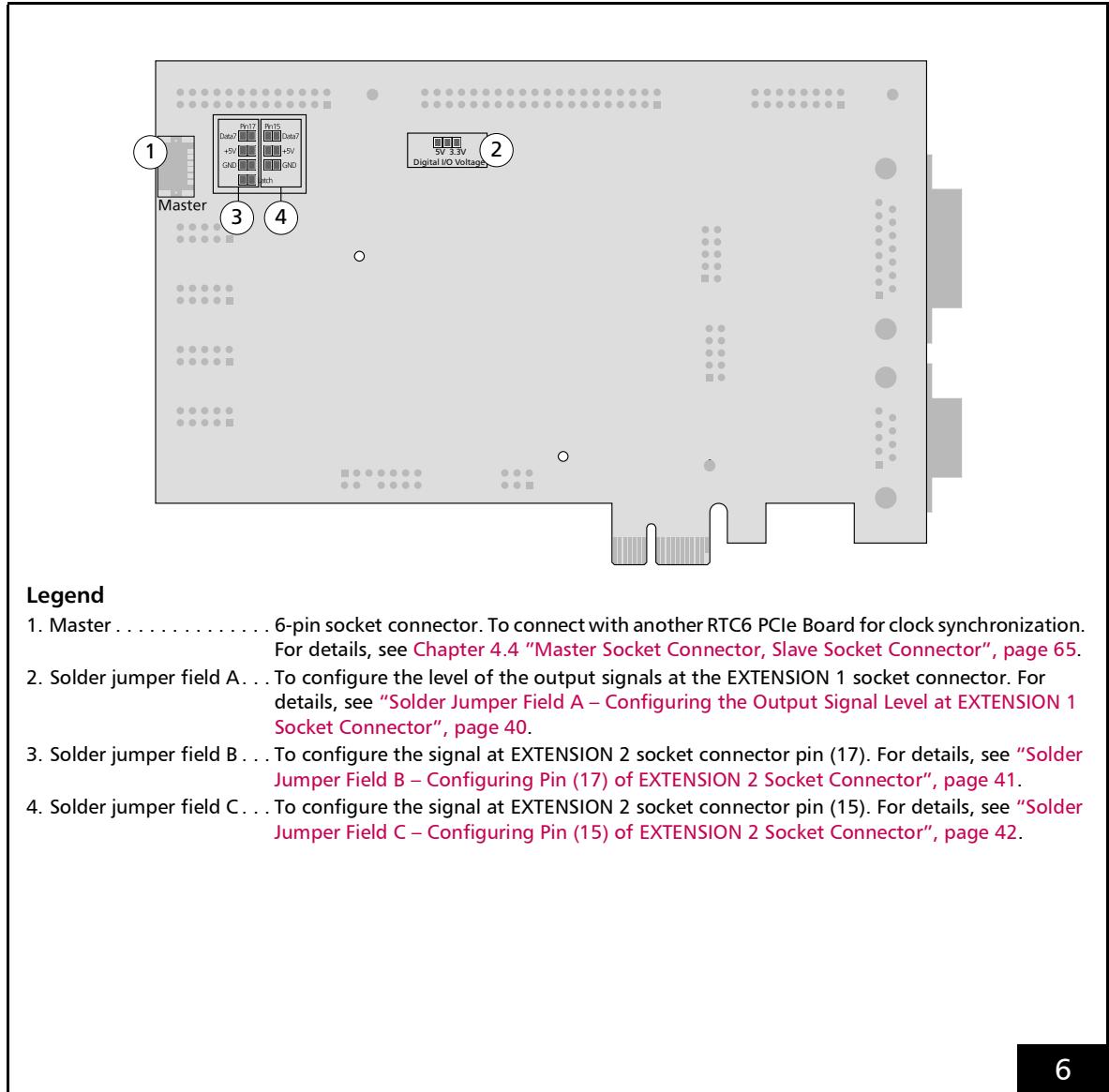
4 RTC6 PCIe Board – Layout and Interfaces

4.1 Layout – Upper Side



RTC6 PCIe Board: upper side.

4.2 Layout – Lower Side



RTC6 PCIe Board: lower side.

4.3 Interface to the PC

The interface to the PC is the PCIe-x-1 connector of the RTC6 PCIe Board, see [Figure 5](#).

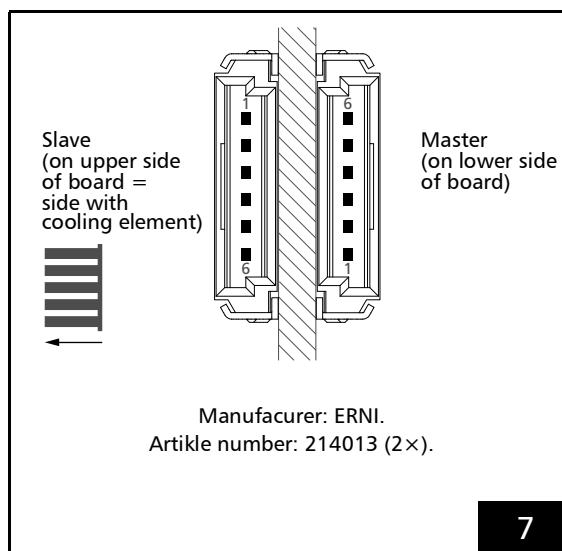
The RTC6 PCIe Board can be installed into any Windows PC with a PCI-Express bus interface and at least one free PCI-Express slot.

Notice!

- The RTC6 PCIe Board does not support power-saving modes that switch off power to the PCIe bus. Accordingly, you must disable standby or sleep modes of the operating system. See also [Section "Notes", page 36](#).

4.4 Master Socket Connector, Slave Socket Connector

The Slave socket connector as well as the Master socket connector have 6 pins, see [Figure 7](#). The Slave socket connector is located on the upper side of the RTC6 PCIe Board, see [Figure 5](#). The Master socket connector is located on the lower side, see [Figure 6^{\(1\)}](#).



7

Master socket connector and Slave socket connector.
The pitch of the pins is 1.27 mm.

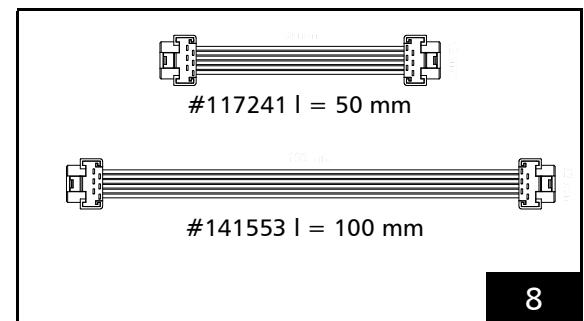
(1) Vice versa with RTC6 Ethernet Boards.

Purpose of the both socket connectors is to make a clock cycle synchronization of several RTC6 Ethernet Boards possible. Then they must be connected pairwise with each other by the Master and Slave socket connectors. Always connect a Master connector of a board to the Slave connector of another board by a suitable cable (available from SCANLAB, see [Figure 8](#)). The necessary information for assembling your own cables is shown in [Figure 9](#).

Interconnected RTC6 PCIe Boards should (recommended) be plugged into adjacent PCIe slots.

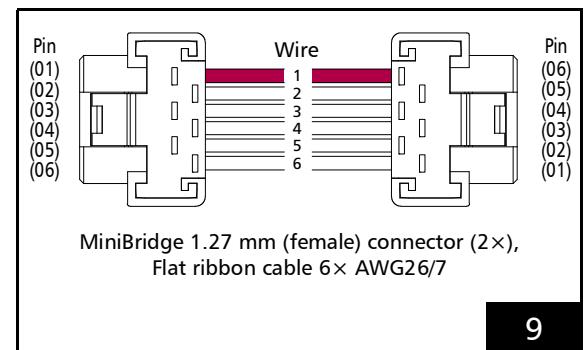
Important: In order to use the master/slave functionality, see prerequisites in [Chapter 6.6.3 "Master/Slave Operation", page 123](#).

See also [Chapter 9.3.1 "Starting and Stopping Lists by External Control Signals and Master/Slave Synchronization", page 288](#).



8

SCANLAB Master/Slave connecting cable.



9

Cable to connect the Master socket connector and Slave socket connector: requirements. Keep length as short as possible!

Notes

- With `master_slave_config` the master/slave interface properties can be configured individually for each RTC6 board.



4.5 Interfaces to Scan System

4.5.1 Scan Head Connectors and Transfer Protocol

The first scan head connector SCANHEAD and the optionally activated second scan head connector 2. SCANHEAD are available for digitally controlling scan systems, see [Figure 10](#).

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 RTC6 PCIe Board and the scan system is in accordance with the SL2-100 protocol. The [XY2-100 Converter \(Accessory\)](#) is available for converting the signals (for data transmission according to the XY2-100 protocol).

If neither the [Option "Second Scan Head Control"](#) nor the [Option "3D"](#) is enabled, only the first scan head connector outputs signals for an xy scan system.

If the [Option "Second Scan Head Control"](#) is enabled, two xy scan systems can be simultaneously controlled by one RTC6 PCIe Board.

If the [Option "3D"](#) is enabled, then a 3-axis scan system can be controlled by the two scan head connectors (if the first scan head connector has been assigned a 3D correction table). Signals can then be outputted by the first scan head connector to an xy scan head – and by both channels of the second scan head connector to the third axis (z axis).

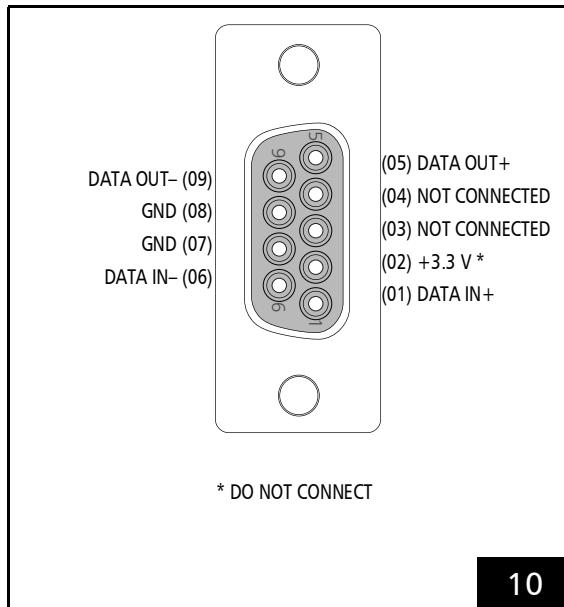
If both options ([Option "Second Scan Head Control"](#) and [Option "3D"](#)) are enabled, the assignment of the correction tables determines which signals (xy or z) are to be outputted by which connector, see also [Section "2D and 3D Correction Files", page 173](#).

If several RTC6 PCIe Boards with enabled [Option "3D"](#) are installed in a PC, then that many 3-axis systems can be simultaneously controlled.

SCANHEAD Connector

(connector for the first scan head)

The pin-out of the first scan head connector SCANHEAD (D-SUB 9-pin female) is shown in Figure 10.



10

SCANHEAD connector (9-pin female D-SUB connector): pin-out.

At the differential DATA OUT output port, the control values for the scan system are outputted.

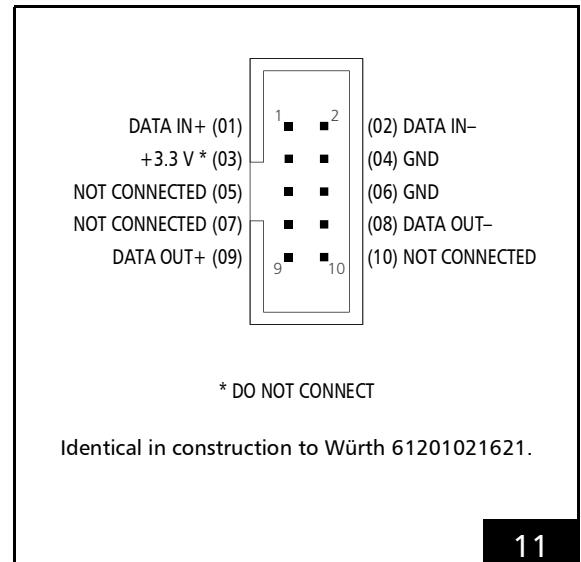
The differential DATA IN input port receives the status signals returned by the scan system.

Pin (02) supplies 3.3 V power for the **XY2-100 Converter (Accessory)** (or a Polymer Optical Fiber converter for optical data transmission). This voltage should not be used for other purposes.

2. SCANHEAD Socket Connector

(connector for the second scan head)

The pin-out of the second scan head connector 2. SCANHEAD (10-pin socket connector) is shown in Figure 11.



Identical in construction to Würth 61201021621.

11

2. SCANHEAD socket connector: pin-out. The pitch of the pins is 2.54 mm. Signals for second scan head are only outputted with enabled **Option "Second Scan Head Control"**.

Second Scan Head Slot Cover (Accessory)

SCANLAB recommends using an additional slot cover for connecting a second scan head or a z axis to the **2. SCANHEAD Socket Connector**, see **Chapter 2.8.4 "Slot Cover with 9-pin D-SUB Connector for "2. SCANHEAD" Socket Connector", page 44.**

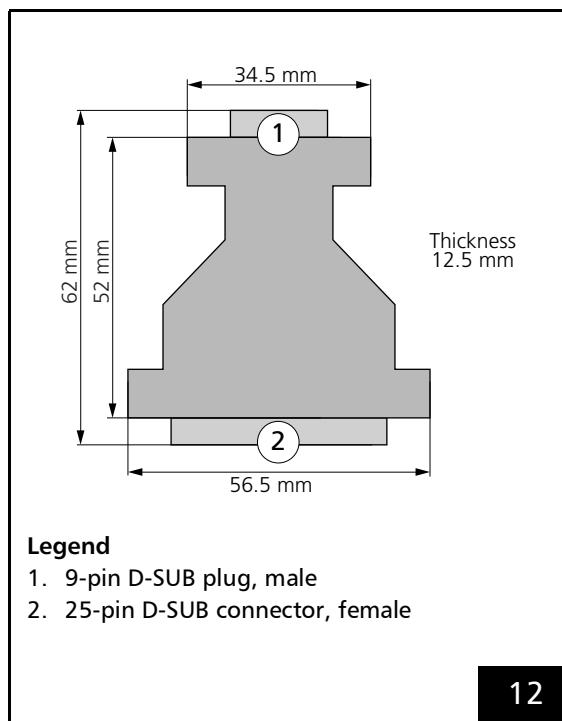
4.5.2 XY2-100 Converter (Accessory)

The SCANLAB **XY2-100 Converter (Accessory)** (#125377) converts:

- SL2-100 protocol-compliant control signals (20 bit) to XY2-100 protocol-compliant control signals (16 bit)
- Scan system XY2-100 protocol-compliant status signals to SL2-100 protocol-compliant status signals, see also **Chapter 7.3.7 "Status Monitoring and Diagnostics"**, page 182

When controlling scan systems, the **XY2-100 Converter (Accessory)** introduces a $10 \mu\text{s}$ signal propagation delay. To compensate for this, the **LaserOn Delay** and **LaserOff Delay** must be increased by $10 \mu\text{s}$ each, see **set_laser_delays**.

The dimensions are shown in **Figure 12**.



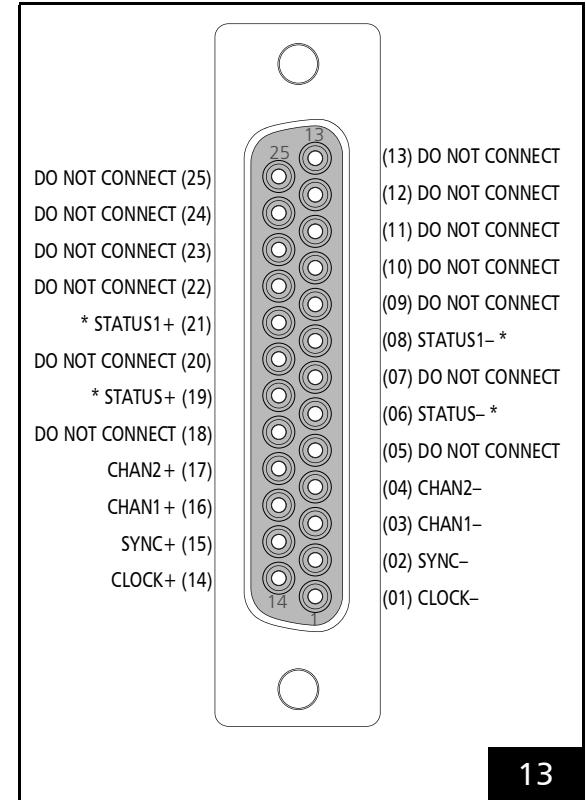
12

XY2-100 Converter (Accessory): dimensions.

The 9-pin D-SUB (male) plug of the **XY2-100 Converter (Accessory)** should be directly plugged into the SCANHEAD connector of the RTC6 PCIe Board or, (by a short-as-possible 1:1 cable) connected to the corresponding 2. SCANHEAD connector pins of the RTC6 PCIe Board. For the pin-out, see **Figure 10** and **Figure 11**.

The 25-pin D-SUB connector (female) of the **XY2-100 Converter (Accessory)** is compatible with scan heads that provide an XY2-100 standard interface.

The pin-out is shown in **Figure 13**.



13

XY2-100 Converter (Accessory): pin-out of the 25-pin D-SUB connector (female).

* For iDRIVE scan systems (see Glossary entry on [page 26](#)), 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".

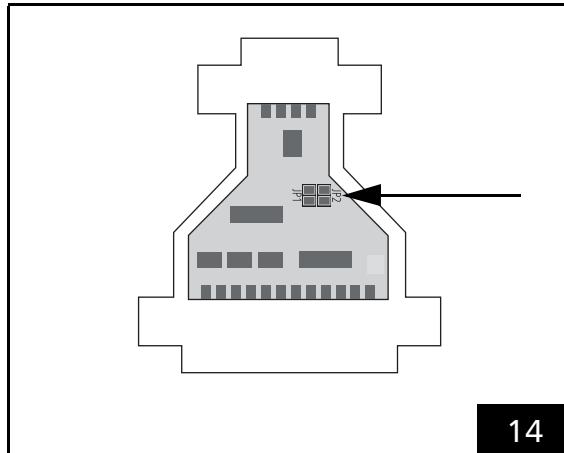
The data channels CHAN1 and CHAN2 transmit control values to the scan head. The SYNC and CLOCK channels transmit synchronization and clock signals to the scan system.

The STATUS channel (and, if appropriate, the STATUS1 channel) receives XY2-100-compliant status signals returned by the scan system.

If cables longer than 20 m (not recommended) are used for data transmission between the **XY2-100 Converter (Accessory)** and the scan system, then the synchronization of bidirectional communication between the scan system and the RTC6 PCIe Board should be configured.

This can be accomplished by two solder jumpers in the **XY2-100 Converter (Accessory)**.

To do so, carefully open the housing of the **XY2-100 Converter (Accessory)** by its 4 clip latches⁽¹⁾. The solder jumpers JP1 and JP2 are on the PCB, see arrow in **Figure 14**.



XY2-100 Converter (Accessory): positions of solder jumpers JP1 and JP2 on the PCB.

The following table shows the possible jumper settings and the corresponding cable lengths. Other jumper settings are not allowed. Cable lengths above 20 m are not recommended.

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

* The cable length range mentioned here for the respective jumper configuration depends on the used cable type and may differ from the values mentioned above. Cable lengths over 20 m are generally not recommended (and require extensive checks by users prior to productive use).

Notes

- Observe the note on [get_head_status](#), page 212.

(1) For example, using a slotted screwdriver.

4.5.3 Data Cables (Accessories)

For transmission of the data signals between the RTC6 PCIe Board (or the [XY2-100 Converter \(Accessory\)](#)) and the scan system, appropriate cables are obtainable from SCANLAB. Data cables are generally not included in the scope of delivery.

For SCANLAB scan systems equipped with an SL2-100 interface (9-pin female D-SUB connector), data cables are available for electrical transmission, for optical transmission (only upon request) also optical fiber cables⁽¹⁾.

For SCANLAB scan systems equipped with a XY2-100 interface (25-pin female D-SUB connector) solely electrical cables are obtainable.

Scan systems equipped with an optical interface (XY2-100-O) cannot be controlled by the RTC6 PCIe Boards.

With self-constructions, SCANLAB recommends the following design for electrical data transmission:

- For SL2-100 protocol-compliant data transmission, see [Figure 15](#), 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 (for example, so that the DATA OUT signal of the RTC6 PCIe Board flows to the DATA IN input of the scan system). 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, see [Figure 16](#), 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 an [XY2-100 Converter \(Accessory\)](#) with standard jumper configuration, the data cable should not be longer than 20 m. If a longer data cable is needed, then the solder jumper setting of the [XY2-100 Converter \(Accessory\)](#) need to be reconfigured.
- For XY2-100-compliant data transmission, the controller end of the data cable must be fitted with a ferrite identical in construction to Würth WE 74271132.

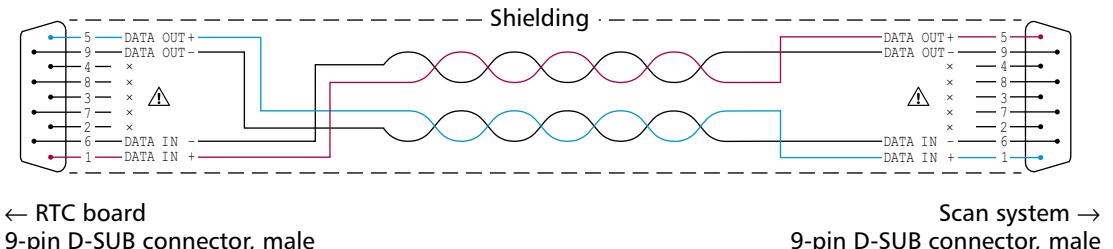
Independently from the data transmission protocol, the following applies in addition:

- 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 *must not* be implemented as a wire. Instead, the cable's braided shielding should be *coaxially* connected to the D-SUB housing by shielded clamps.

Some *scan heads* have only a single connector to provide both the operating voltage and the data signals. For these *scan heads*, SCANLAB recommends implementing a cabling solution which uses a separate cable for data. The cable should have the properties as already described, see above.

(1) The optical fiber cables, too, are attached by 9-pin D-SUB connectors. Optical conversion (Polymer Optical Fiber conversion) for optical data transmission takes place inside the D-SUB connectors. The operating voltage for Polymer Optical Fiber conversion is supplied at the RTC6 PCIe Board scan head connectors and the digital interface of the scan system.

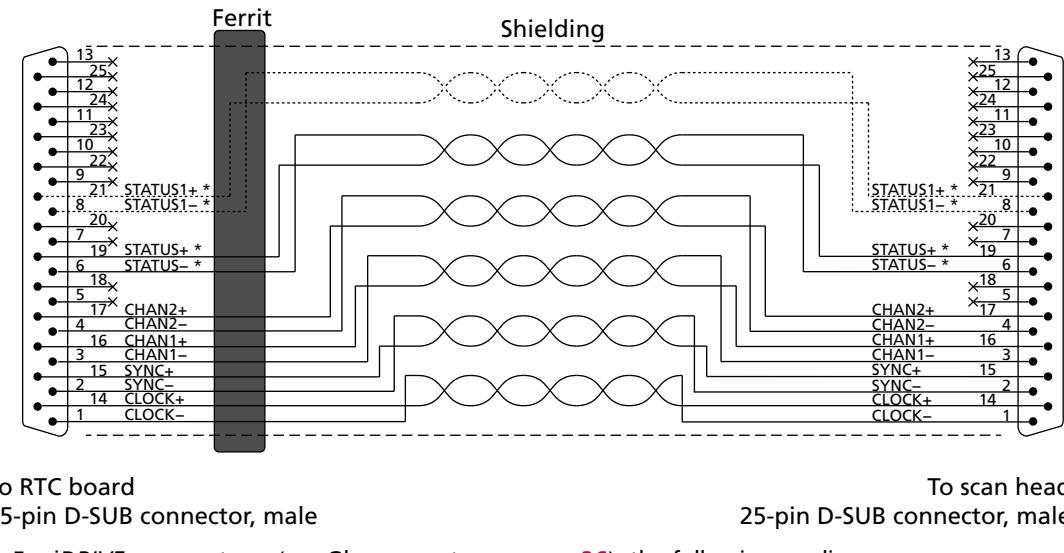
Cable for SL2-100 protocol-compliant data transmission



15

Cable for data transmission compliant to SL2-100 protocol: requirements and pin assignments:
requirements and pin assignments.

Cable for XY2-100 protocol-compliant data transmission



- * For iDRIVE scan systems (see Glossary entry on [page 26](#)), the following applies:
 - The STATUS \pm channel is the status channel for axis 2 (x axis; this channel is also called STATUS2 \pm there).
 - The STATUS1 \pm channel is the status channel for axis 1 (y axis).
 - For other scan systems, the STATUS1 \pm channel is not needed.

16

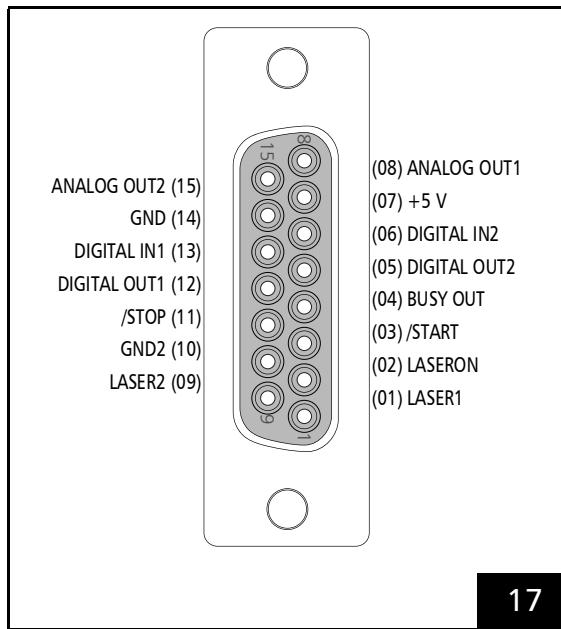
Cable for data transmission compliant to XY2-100 protocol:
requirements and pin assignments.

4.6 Interfaces for the Laser and Peripheral Equipment

4.6.1 LASER Connector

The **LASER Connector** is a 15-pin D-SUB connector (female). It is located on the slot cover of the RTC6 PCIe Board, see [Figure 5](#).

The pin-out is shown in [Figure 17](#).



17

LASER Connector (15-pin D-SUB connector, female): pin-out. On RTC6 PCIe Boards without [Option "DC/DC Converter"](#), GND2 are GND identical.

Notice!

- If you want to use the RTC6 PCIe Board in conjunction with laserDESK, observe the following:
 - laserDESK uses additional pins on the **EXTENSION 1 Socket Connector**, for example, to control the laser.
 - Refer to the extended documentation for connecting the laser. Refer to laserDESK online help (alternatively available from SCANLAB or in laserDESK-zip, which can be downloaded from the SCANLAB website).

Laser Control Signals

The laser control signals are LASERON, LASER1 and LASER2. They are digital TTL level signals and are referenced to GND2.

The laser control signals LASER1 and LASER2 differ depending on the set laser mode, see following table and the timing diagrams in [Chapter 7.4 "Laser Control", page 183](#).

	LASER1 pin (01)	LASER2 pin (09)
CO₂ Mode	Modulation pulse 1, standby signal	Modulation pulse 2, standby signal
YAG Modes 1, 2, 3, 5	Q-Switch signal	FirstPulseKiller signal
Laser Mode 4	Standby signal	FirstPulseKiller signal
Laser Mode 6	Standby signal	–

On RTC6 PCIe Boards with [Option "DC/DC Converter"](#), GND2 and the laser control signals are galvanically decoupled from GND⁽¹⁾. On RTC6 PCIe Boards without [Option "DC/DC Converter"](#), GND2 are GND identical, see [Chapter 2.6 "Options", page 37](#).

For the maximum current load see [Chapter 15 "Technical Specifications – RTC6 PCIe Board", page 855](#).

All laser control signals can be set by **set_laser_control** to either active-LOW or active-HIGH logic. "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 control signal level according to the specifications of your laser control. Observe the documentation of your laser.

Pin (01), (02) and (09) of the **LASER Connector** can be configured by **config_laser_signals** and **config_laser_signals_list**, see also [Chapter 7.4.2 "Configuring the LASER Connector", page 186](#).

(1) With RTC6 PCIe Boards GND is the PC ground. With RTC6 Ethernet Boards GND is the ground at the POWER connector.

External Control Signals

The external control signals are /START and /STOP (TTL active-LOW). Both input signals are connected internally to +3.3 V by pull-up resistors (4,7 k Ω), see [Figure 18](#). The signals are referenced to GND⁽¹⁾. See also [Chapter 9.3.1 "Starting and Stopping Lists by External Control Signals and Master/Slave Synchronization", page 288](#).

BUSY List Execution Status

The **BUSY list execution status** is available as the **BUSY OUT** signal at pin (04). When the **BUSY list execution status** is set, the **BUSY OUT** signal is HIGH. See also [Chapter 6.4.3 "List Execution Status", page 107](#). The signal is referenced to GND⁽¹⁾.

2-Bit Digital Input Port

The RTC6 PCIe Board provides a 2-bit digital input port (DIGITAL IN1 and DIGITAL IN2).

For technical data see [Chapter 15 "Technical Specifications – RTC6 PCIe Board", page 855](#).

For programming the input port see [Chapter 9.2.2 "2-Bit Digital Input Port", page 287](#).

2-Bit Digital Output Port

The RTC6 PCIe Board provides a buffered 2-bit digital output (DIGITAL OUT1 and DIGITAL OUT2).

For technical data see [Chapter 15 "Technical Specifications – RTC6 PCIe Board", page 855](#).

For programming the output port see [Chapter 9.1.3 "2 Bit Digital Output Port", page 282](#).

12-Bit Analog Output Port 1 and 2

The RTC6 PCIe Board provides two 12-bit analog output ports:

- ANALOG OUT1
- ANALOG OUT2⁽²⁾

For technical data see [Chapter 15 "Technical Specifications – RTC6 PCIe Board", page 855](#).

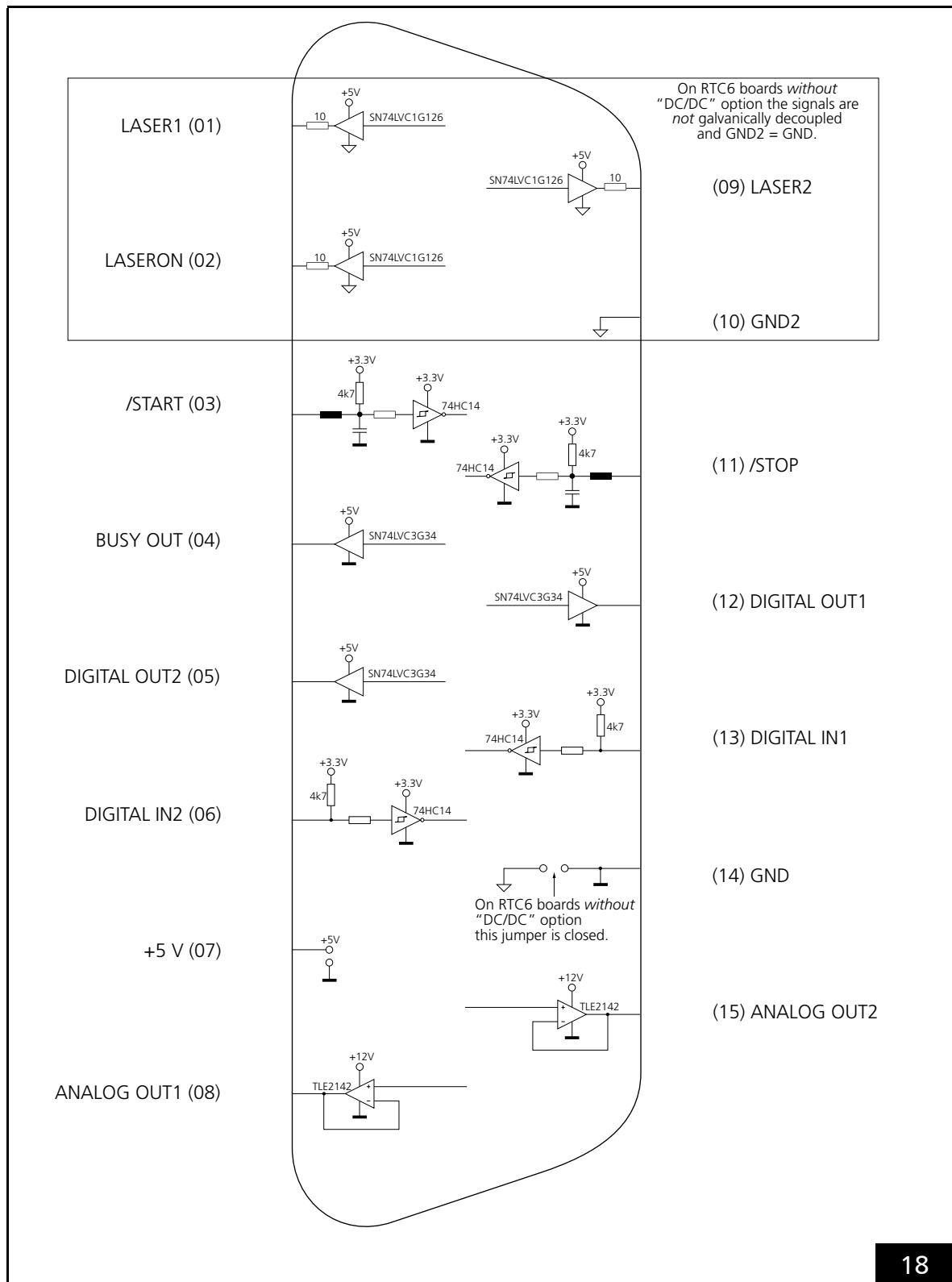
For programming the output ports, see [Chapter 9.1.4 "12-Bit Analog Output Port 1 and 2", page 282](#).

Input and Output Wiring

The input and output wiring of the **LASER Connector** is shown in [Figure 18](#).

(2) The signal of **ANALOG OUT2** is also available by the **MARKING ON THE FLY** socket connector, see [Section "Analog Output Port", page 81](#).

(1) See footnote on [page 72](#).



LASER Connector, see also Figure 17: input/output wiring.
See also Section "Option "DC/DC Converter"" , page 37.

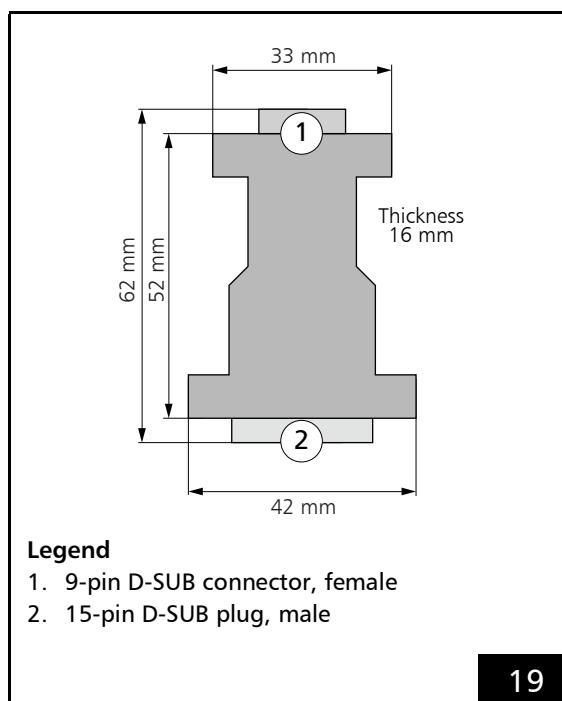
Laser Adapter (Accessory)

The SCANLAB laser adapter (accessory; #114774) is plugged into the 15-pin **LASER Connector** of the RTC6 PCIe Board. Then its 9-pin D-SUB connector (female) provides the same signals and pin-out as the RTC4 9-pin LASER connector.

Notes

- The laser adapter can *not* be directly plugged into the RTC6 PCIe Board if an **XY2-100 Converter (Accessory)** is already plugged in.
- The **BUSY OUT** signal, 2-bit digital input port and 2-bit digital output port are *not* available at the laser adapter's 9-pin D-SUB connector.

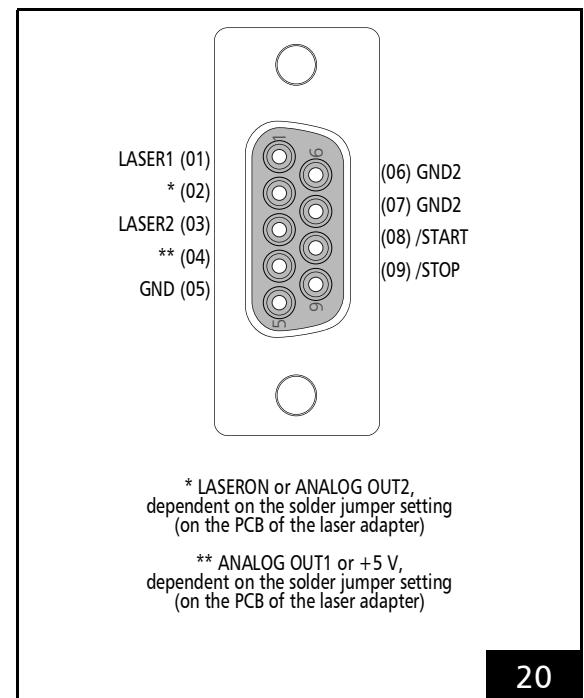
The dimensions of the laser adapter is shown in **Figure 19**.



19

Laser adapter (accessory): dimensions.

The pin-out of the 9-pin D-SUB connector is shown in **Figure 20**.

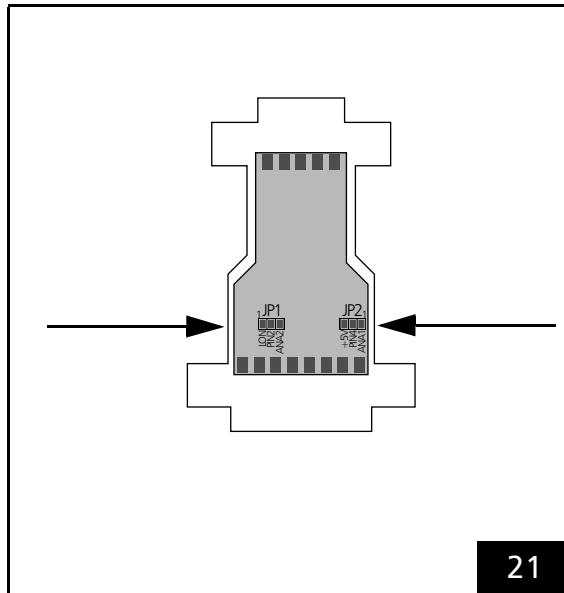


20

Laser adapter (accessory): pin-out of the 9-pin D-SUB connector, female.

The signals at pins (02) and (04) 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 housing by its 4 clip latches (for example, using a screwdriver). The solder jumpers **JP1** and **JP2** are on the PCB of the laser adapter between the two D-SUB connectors.

The positions of the solder jumpers on the PCB is shown in [Figure 21](#).



21

Laser adapter (accessory): position of solder jumper **JP1** and **JP2** on the printed circuit board.

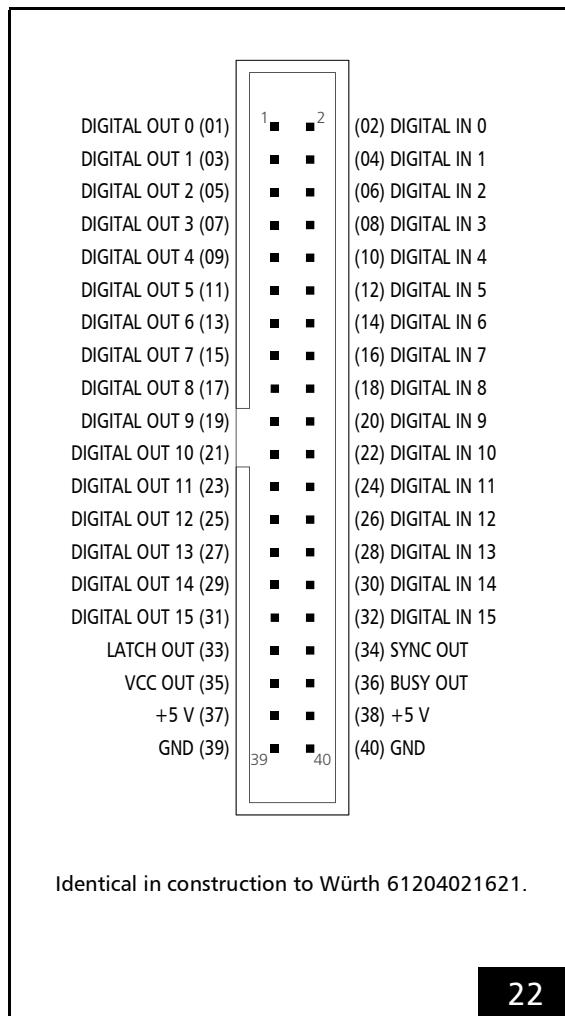
The following table shows the possible jumper settings. Other jumper settings are not allowed.

Solder jumper JP1: signal at pin (02)	Position 1-2 1 PIN2 ANA2 LASERON (default configuration)	Position 2-3 1 PIN2 ANA2 ANALOG OUT2
Solder jumper JP2: signal at pin (04)	Position 1-2 1 +5V PIN4 ANA1 ANALOG OUT1 (default configuration)	Position 2-3 1 +5V PIN4 ANA1 +5 V

4.6.2 EXTENSION 1 Socket Connector

The EXTENSION 1 socket connector has 40 pins. It is located on the upper side of the RTC6 PCIe Board, see [Figure 5](#).

The pin-out is shown in [Figure 22](#).



Configuring the Output Signal Level

With the solder jumper field A on the lower side of the RTC6 PCIe Board, see [Figure 6](#), the level of all output signals at the EXTENSION 1 socket connector (DIGITAL OUT 0...DIGITAL OUT 15, LATCH_OUT, SYNC_OUT, BUSY_OUT, VCC_OUT) can be configured for 5 V or 3.3 V, see [Chapter 2.7.1](#)

["Solder Jumper Field A – Configuring the Output Signal Level at EXTENSION 1 Socket Connector", page 40](#).

For monitoring purposes, the selected signal level is continuously outputted at pin (35): signal VCC_OUT. VCC_OUT is referenced to GND⁽¹⁾.

The maximum current load of the signal is 100 mA.

16-Bit Digital Input Port and 16-Bit Digital Output Port

The 40-pin EXTENSION 1 socket connector provides a 16-bit digital TTL input and a buffered 16-bit digital TTL output, see [Figure 22](#). This requires the output signals level to be configured by the jumper setting, see [Section "Configuring the Output Signal Level", page 77](#).

For programming, see

- [Chapter 9.1.1 "16-Bit Digital Output Port", page 281](#)
- [Chapter 9.2.1 "16-Bit Digital Input Port", page 287](#)
- [Chapter 9.3.2 "Conditional Command Execution", page 294](#)

The input and output signals are referenced to GND⁽¹⁾.

The maximum current load of the output signals is 8 mA.

22

EXTENSION 1 socket connector: pin-out. The pitch of the pins is 2.54 mm.

(1) See footnote on [page 72](#).

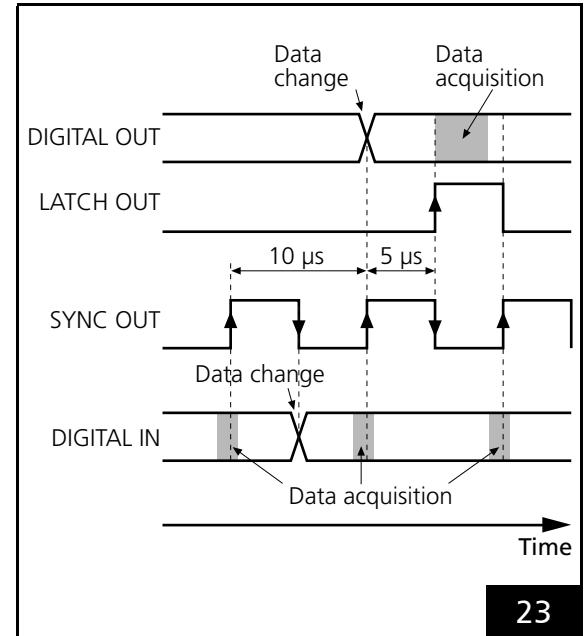
Synchronization of Data Acquisition

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

The LATCH OUT signal (a 5 μ s 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 RTC6 PCIe Board automatically generates the LATCH OUT signal when the output value at the 16-bit digital output is written. The output value should be read-out with the rising edge of the LATCH OUT signal. The rising edge occurs 5 μ s after the value has been outputted at the 16-bit digital output, see [Figure 23](#).

To synchronize data acquisition at the 16-bit digital input port, a SYNC OUT signal (a square wave with 5 μ s pulse length and 10 μ s period) at pin (34) is continuously outputted. Value changes at the 16-bit digital input port should be made with a falling edge of the SYNC OUT signal. The DSP of the RTC6 PCIe Board always accepts the currently provided value with the rising edge of the SYNC OUT signal, see [Figure 23](#).

The synchronization signals LATCH OUT and SYNC OUT are referenced to GND⁽¹⁾. The maximum current load is 10 mA.



23

Synchronization of data acquisition by LATCH OUT signal or SYNC OUT signal.

BUSY List Execution Status

The BUSY OUT signal at pin (36) is identical to the BUSY OUT signal at the **LASER Connector**⁽²⁾, see [Section "BUSY List Execution Status", page 73](#). The signal is referenced to GND⁽¹⁾.

(1) See footnote on [page 72](#).

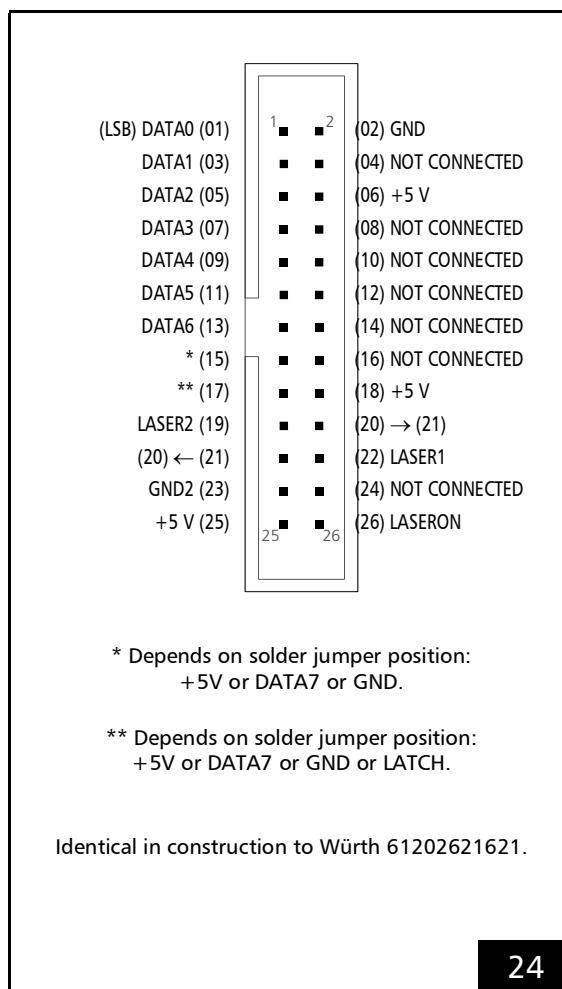
(2) RTC6 Ethernet Board: **LASER** socket connector.

4.6.3 EXTENSION 2 Socket Connector

The EXTENSION 2 socket connector⁽¹⁾⁽²⁾ has 26 pins. It is located on the upper side of the RTC6 PCIe Board, see [Figure 5](#).

It provides a buffered 8-bit digital output port: DATA0...DATA7.

The pin-out is shown in [Figure 24](#).



24

EXTENSION 2 socket connector: pin-out. The pitch of the pins is 2.54 mm.

Notes

- Pin (15) and pin (17) are configurable by solder jumpers.

- (1) On the RTC4, the functional corresponding socket connector is labeled LASER EXTENSION.
- (2) RTC6 Ethernet Boards do not provide laser control signals (LASER1 and LASER2) at their (10-pin) EXT. 2 socket connectors.

Configuration by Solder Jumpers

Pin (15) is configured by the solder jumper field C, see [Chapter 2.7.3 "Solder Jumper Field C – Configuring Pin \(15\) of EXTENSION 2 Socket Connector", page 42](#).

Pin (17) is configured by solder jumper field B, see [Chapter 2.7.2 "Solder Jumper Field B – Configuring Pin \(17\) of EXTENSION 2 Socket Connector", page 41](#).

Notes

- If the DATA7 bit is assigned to pin (15), then the full 8-bit output value is available at the output port (at the odd numbered pins of the EXTENSION 2 socket connector, pin (01)...pin (15)).
- If pin (15) is set to +5 V (HIGH level), then the output values have an offset of 128. That is, the output values are between 128...255.
- If pin (15) is set to GND (LOW level), then the possible output values are 0...127.
- The DATA7 bit can be used for other purposes by assigning it to pin (17).

Laser Control Signals

The laser control signals LASER1 and LASER2 are identical to the [LASER Connector](#) and depend on the set laser mode, see [Section "Laser Control Signals", page 72](#).

The signals are referenced (as with the [LASER Connector](#)) to GND2.

On RTC6 PCIe Boards with [Option "DC/DC Converter"](#), GND2 and the laser control signals are galvanically decoupled from GND⁽¹⁾.

On RTC6 PCIe Boards without [Option "DC/DC Converter"](#), GND2 are GND identical, see [Chapter 2.6 "Options", page 37](#).

8-Bit Digital Output Port

The buffered 8-bit digital output port (TTL level) is intended for lasers with digital power control (for example, to control the lamp current of a YAG laser).

Of course, it can be used for any other purpose as well. The output is in high-impedance mode until an initial value is assigned to it.

For programming the output see [Chapter 9.1.2 "8-Bit Digital Output Port", page 282](#).

The **MSB** (DATA7) of the output value can be used for other purposes. To do this, it can be assigned to an other pin on the EXTENSION 2 socket connector (pin (15), pin (17), see above).

If the solder jumper is correspondingly set (see above), Pin (17) outputs a LATCH signal. The RTC6 PCIe Board automatically generates the LATCH signal (a 5 µs pulse, active-HIGH) when the value at the 8-bit digital output port is outputted.

If several bits are simultaneously transferred as a data word (and not independently from each other) by the 8-bit digital output port, then the LATCH signal should be used for synchronization of data transmission.

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 is written at the 8-bit digital output port, see also [Figure 23](#).

The LATCH signal is referenced to GND⁽¹⁾. Its maximum current load is 10 mA.

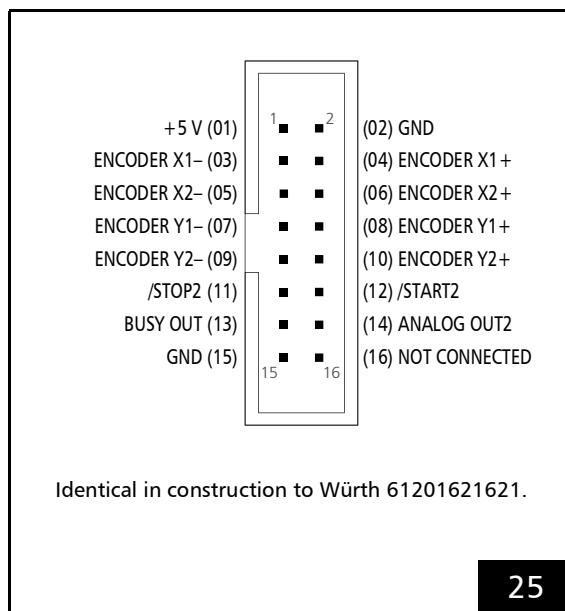
(1) See footnote on [page 72](#).

4.6.4 MARKING ON THE FLY Socket Connector

The MARKING ON THE FLY socket connector has 16 pins and is located on the upper side of the RTC6 PCIe Board, see [Figure 5](#). It provides, among other things, pins for encoder inputs.

If the [Option Processing-on-the-fly](#) of the board is enabled, laser material processing of moving workpieces (for example, on a moving conveyor belt or rotating disk) can be implemented, see [Chapter 8.6 "Processing-on-the-fly", page 241](#).

The pin-out is shown in [Figure 25](#).



MARKING ON THE FLY socket connector: pin-out. The pitch of the pins is 2.54 mm.

Encoder Input Ports

The RTC6 PCIe Board provides two encoder input ports:

- ENCODER X
- ENCODER Y

Each of the encoder input ports is designed for a pair (1, 2) of standardized RS-422 differential signals. See also [Section "Input Ports for External Encoder Signals", page 298](#).

External Control Signals

The external control signals /START2 and /STOP2 (TTL active-LOW) are referenced to GND⁽¹⁾. 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 288](#).

Analog Output Port

The signal outputted at the 12-bit analog output port **ANALOG OUT2** of the [LASER Connector](#) is also outputted at pin (14), see also [Section "12-Bit Analog Output Port 1 and 2", page 73](#).

The signal is referenced to GND⁽¹⁾.

BUSY List Execution Status

The BUSY OUT signal at pin (13) is identical to the BUSY OUT signal at the [LASER Connector](#), see [Section "BUSY List Execution Status", page 73](#).

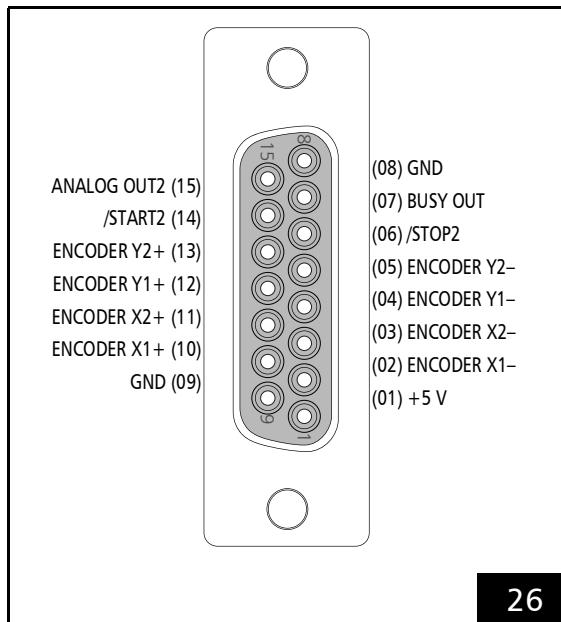
The signal is referenced to GND⁽¹⁾.

(1) See footnote on [page 72](#).

MARKING ON THE FLY Slot Cover (Accessory)

SCANLAB recommends using an additional slot cover for using the input ports and signals of the **MARKING ON THE FLY Socket Connector**, see [Chapter 2.8.5 "Slot Cover with 15-pin D-SUB Connector for "MARKING ON THE FLY" Socket Connector", page 44](#).

The pin-out is shown in [Figure 26](#).



MARKING ON THE FLY Slot Cover (Accessory), 109272: pin-out of the 15-pin female D-SUB connector.

Alternatively, the [Slot Cover with 15-pin D-SUB Connector and 9-pin D-SUB Connector, page 45](#) is also available.

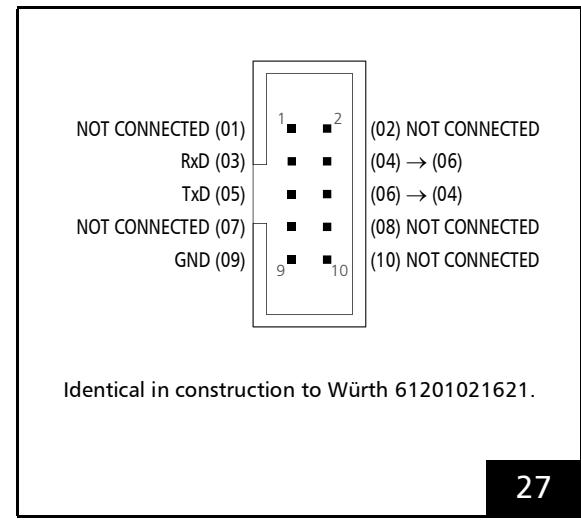
4.6.5 RS232 Socket Connector

The RS232 socket connector has 10 pins and is located on the upper side of the RTC6 PCIe Board, see [Figure 5](#).

The RS232 socket connector is a hardware interface designed for bidirectional data exchange.

By [uart_config](#) (as of DLL 612), the RS-232 interface can be configured⁽¹⁾. [uart_config](#) returns the actual set Baud rate.

The pin-out is shown in [Figure 27](#).



RS232 socket connector: pin-out. The pitch of the pins is 2.54 mm.

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

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

All signals are referenced to GND⁽²⁾.

The baud rate (default: 9600 baud) can be set with [uart_config](#) (to 160...12,800,000 Baud). Other parameters cannot be altered (data bits: 8, start bits: 1, stop bits: 1, parity: none).

For outputting data by the RS-232 interface, see [Chapter 9.1.6 "RS-232 Interface", page 286](#).

For reading-in data by the RS-232 interface, see [Chapter 9.2.3 "RS-232 Interface", page 287](#).

(1) Alternatively, the older [rs232_config](#) can be used which does not return a Baud rate. With < DLL 612, the maximum Baud rate is limited to the range [300...115200].

(2) See footnote on [page 72](#).

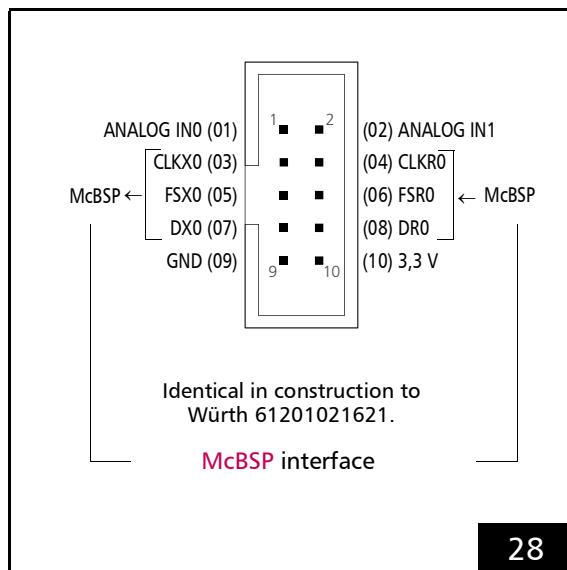
4.6.6 McBSP/ANALOG Socket Connector

The McBSP/ANALOG socket connector⁽¹⁾ has 10 pins. It is located on the upper side of the RTC6 PCIe Board, see [Figure 5](#).

The McBSP/ANALOG socket connector is a hardware interface that

- is designed for bidirectional data exchange (McBSP), see [Section "McBSP Interface", page 83](#) and
- furthermore, provides two 10 V analog input ports: ANALOG IN0 at pin (01) and ANALOG IN1 at pin (02), see [Section "Analog Input Ports", page 85](#).

The pin-out is shown in [Figure 28](#).



28

McBSP/ANALOG socket connector: pin-out. The pitch of the pins is 2.54 mm.

McBSP Interface

The [McBSP interface](#), see [Figure 28](#), is initialized by [load_program_file](#) with:

- `XDelay = RDelay = 1`
- 8 MHz clock frequency

Intended for [McBSP](#) are pin (03), (05), (07) (outgoing signals) and pin (04), (06), (08) (incoming signals).

The following signals are continuously outputted after [load_program_file](#):

- the clock signal at pin (03)
- every 10 μ s a frame synchronization signal FSX at pin (05)
- a data signal DX at pin (07) (Default: `SampleY|SampleX`), see [set_mcbsp_out](#)

The [McBSP interface](#) can be integrated into applications for various purposes:

- As an alternative to encoder signals, the position of a moving workpiece can be directly integrated, see also [Chapter "Correction via McBSP Interface", page 244](#). Prerequisite: [Option Processing-on-the-fly](#).
- As an alternative to matrix and offset commands, coordinate transformations can be transmitted and integrated to align a workpiece with controllable timing ([Online Positioning](#)), see also [Chapter 9.3.4 "Synchronization and Online Positioning by McBSP Signals", page 299](#).
- With [set_multi_mcbsp_in](#) you can transfer not only a 3D fly correction with position information but also laser power and other parameters, see also [Chapter "Correction via McBSP Interface with Additional McBSP Input", page 245](#). Prerequisite: [Option Processing-on-the-fly](#).
- Permanent data output in 10 μ s cycles. With [set_mcbsp_out](#) and [set_mcbsp_out_ptr](#) it can be selected which data signals are outputted.
- [OIE Output Mode, page 687](#) can be set at the McBSP interface by [set_mcbsp_out_oie_ctrl](#) and [set_mcbsp_out_oie_list](#).

(1) With RTC5 boards referred to as [SPI /I2C](#) socket connector.

For data output using the **McBSP interface**, see **Chapter 9.1.7 "McBSP Interface"**, page 286.

For data input using the **McBSP interface**, see **Chapter 9.2.4 "McBSP Interface"**, page 287.

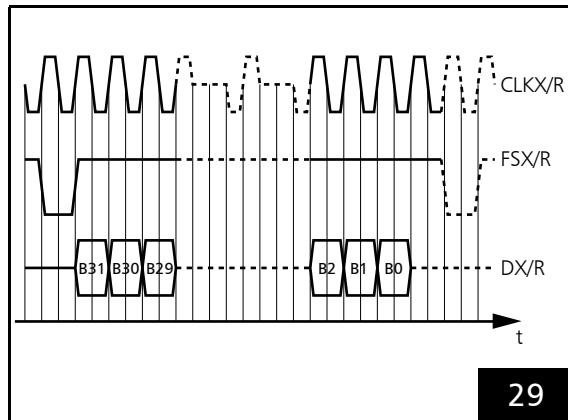
mcbsp_init allows setting a data delay for the **McBSP** data transmission independently for the transmitter and receiver. Possible values range from 0 to 2 (default: 1).

All signals are referenced to GND⁽¹⁾.

RTC6 PCIe Board 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
 - Data delay **XDelay** bits
- The timing diagram of the **McBSP** signals is shown in **Figure 29**. 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 Bit #31...Bit #0. The transmission frequency is by default 8 MHz (4 μ s per data word). Alternatively, a value between 4 MHz and 16 MHz can be set by **set_mcbsp_freq**.



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

RTC6 PCIe Board as Receiver

The following specifications apply to CLKR0, FSR0, DR0:

- Signal level 3.3 V or 5 V TTL.
- **McBSP mode:**
 - Single phase frame
 - Single element per frame
 - 32 bits per element
 - Data delay **RDelay** bits

The timing diagram of the **McBSP** signals is shown in **Figure 29**. 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.

When doing so, observe the following Notes:

- The **McBSP interface** always ignores the first frame synchronization signal after a **load_program_file** or **mcbsp_init**. Therefore, the data provided is not transmitted. If necessary, a dummy value should be initially sent to the interface. You can use **read_mcbsp** to check for successful transmission.
- 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.

(1) See footnote on page 72.

- 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 `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 an `Online Positioning` the input values are stored to internal memory locations 1 or 2
 - For a “`Local Online Positioning`” by `set_mcbsp_x`, `set_mcbsp_y` and/or `set_mcbsp_rot` or `set_mcbsp_matrix`, see Section “Configuring “`Local Online Positioning`”, page 228
 - For a “`Global Online Positioning`” by `set_mcbsp_global_x`, `set_mcbsp_global_y` and/or `set_mcbsp_global_rot` or `set_mcbsp_global_matrix`, see Section “Configuring “`Global Online Positioning`”, page 230
 - The most recent fully transferred values can be queried from a corresponding memory location by `read_mcbsp`.
 - 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.
 - 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.

Analog Input Ports

The McBSP/ANALOG socket connector (RTC6 Ethernet Boards: SPI/ANA/UART socket connector) provides two analog input ports⁽¹⁾:

- ANALOG IN0
- ANALOG IN1

After the first `read_analog_in` call, voltages that are applied there are:

- automatically converted endlessly (duration approx. 0.3 ms)⁽²⁾
- transferred to the DSP as 12-bit digital values

By the control command `read_analog_in`, the present values can be queried at any time.

For technical specifications, see [Analog input ports, page 857](#).

(1) As with RTC5 PCI Express Boards (but not with RTC5 PCI Boards).

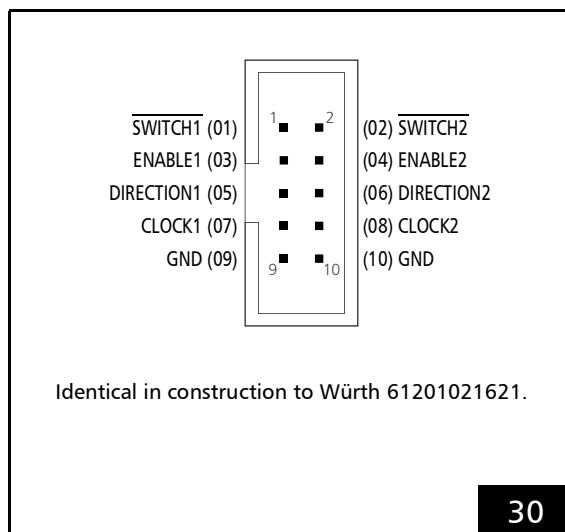
(2) Note the comment with `get_temperature`, page 428.

4.6.7 STEPPER MOTOR Socket Connector

The STEPPER MOTOR socket connector has 10 pins and is located on the upper side of the RTC6 PCIe Board, see [Figure 5](#).

At the STEPPER MOTOR socket connector, signals for controlling up to two stepper motors can be outputted.

The pin-out is shown in [Figure 30](#).



STEPPER MOTOR socket connector: pin-out. The pitch of the pins is 2.54 mm.

All signals are referenced to GND⁽¹⁾.

The output signals (ENABLE, DIRECTION and CLOCK) are TTL level signals (5 V).

For each step that is to be carried-out the RTC6 PCIe Board generates an active-HIGH 5 μ s pulse as CLOCK signal. 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 two SWITCH input ports serve to connect end switches. The RTC6 PCIe Board detects an end switch position by a LOW level (active-LOW). Internally the SWITCH input ports are connected to +3.3 V by 10 k Ω pull-up resistors.

For programming the stepper motor signals, see [Chapter 9.1.5 "Controlling Stepper Motors", page 283](#).

(1) See footnote on [page 72](#).

5 Installation and Start-Up

Installation of the RTC6 PCIe Board consists of the following steps:

- (1) Checking the RTC6 PCIe Board jumpers, see [Chapter 5.1 "Checking Jumper Settings", page 87](#)
- (2) Hardware installation: Installing the RTC6 PCIe Board, see [Chapter 5.2 "Installing the RTC6 PCIe Board", page 87](#)
- (3) Installing the RTC6 board driver, see [Chapter 5.3 "Installing the RTC6 Board Driver", page 88^{\(1\)}](#)
- (4) Installing the RTC6 software, see [Chapter 5.4 "Installing the RTC6 Software", page 89](#)
- (5) When turning on the laser system, observe the safe start-up sequence, see [Chapter 5.5 "Safe Start-up and Shutdown Sequences", page 90](#)
- (6) For conducting a post-installation functionality test, for example, the included HPGL converter program can be used, see [Chapter 5.6 "Functionality Test", page 90](#)

5.1 Checking Jumper Settings

SCANLAB ships RTC6 PCIe Boards in various jumper configurations.

- (1) Make sure that the to-be-installed RTC6 PCIe Board has the required jumper configuration. If you need to change the factory settings, proceed as described in [Chapter 2.7 "Jumper Settings and Type Designations", page 39](#).
- (2) Proceed with [Chapter 5.2 "Installing the RTC6 PCIe Board", page 87](#).

5.2 Installing the RTC6 PCIe Board

Notice!

- Store the board 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 board.
- Protect the board from humidity, dust, corrosive vapors and mechanical stress.

Perform the following steps to install the RTC6 PCIe Board in a PC:

- (1) Remove all data media from your PC.
- (2) Shut down the operating system and switch off the PC. Disconnect the PC from the mains.
- (3) Disconnect all peripherals (for example, 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 PCIe slot and remove the slot cover. A height from the slot connector's top of at least 105 mm is required.
- (6) Remove the RTC6 PCIe Board from the antistatic bag. Do not touch the contacts of the board.

(1) Step 3 does not apply to RTC6 Ethernet Boards.

- (7) If you want to use the signals on the RTC6 PCIe Board socket connectors, then attach the appropriate cables. SCANLAB recommends installing additional slot covers with suitable connectors. Then the signals are accessible even when the PC housing is closed. All RTC6 PCIe Board connectors and signals are described in [Chapter 4 "RTC6 PCIe Board – Layout and Interfaces", page 63.](#)
- (8) Install the RTC6 PCIe Board into the PCIe slot. Observe the instructions in the manual of your PC⁽¹⁾.
- (9) Close the PC housing.
- (10) Place the PC at its operational location and connect all peripheral equipment.
- (11) Connect the 9-pin D-SUB connector on the RTC6 PCIe Board – using the [XY2-100 Converter \(Accessory\)](#), if necessary – to the scan system by a data cable, see [Chapter 4.5 "Interfaces to Scan System", page 66.](#)
- (12) Connect the 15-pin D-SUB connector on the RTC6 PCIe Board to the laser by a suitable interface, see [Chapter 4.6 "Interfaces for the Laser and Peripheral Equipment", page 72.](#)
- (13) If necessary, connect one or more of the RTC6 PCIe Board 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 several master/slave-synchronized RTC6 PCIe Boards are to be used in a PC, then the boards must be connected pairwise with each other by the Master and Slave connectors and installed in preferably (recommended) adjacent PCIe slots, see also [Chapter 4.4 "Master Socket Connector, Slave Socket Connector", page 65.](#)

5.3 Installing the RTC6 Board Driver

Notice!

- If on your PC an WDM technology-based RTC3/4/5 board driver
 - is yet installed or
 - was installed and has been removed or
 - you are not sure in this regard:
 - (1) Install the RTC6 board driver.
 - (2) Run `ScanlabClassChecker.cmd` as Administrator (part of the delivered software package; see there also the background information in [ReadMe_ScanlabClassChecker.pdf](#)). Step 2 can be skipped on brand new PCs on which an RTC board driver never has been installed.

To install the RTC6 board driver for Windows 10, 8, 7, proceed as follows:

- After the RTC6 PCIe Board has been installed, start the computer.
- For Windows 7:
- (1) If the "Add Hardware Wizard" does not come up automatically, call it from the Control Panel.
 - (2) In the "Add Hardware Wizard" specify the folder that includes the RTC6 board driver. During installation, the operating system automatically selects the appropriate driver file (32-bit or 64-bit).
 - (3) Run `ScanlabClassChecker.cmd` as Administrator (part of the delivered RTC6 Software Package; see there also the background information in [ReadMe_ScanlabClassChecker.pdf](#)).

For Windows 10, 8:

- (1) Open the Device Manager to display the device tree. Look for the "PCI device" entry and update its driver by specifying the folder that includes the RTC6 board driver.
- (2) Run `ScanlabClassChecker.cmd` as Administrator (part of the delivered software package; see there also the background information in [ReadMe_ScanlabClassChecker.pdf](#)).

Notice!

- The RTC6 PCIe Board does not support power-saving modes that switch off power to the PCIe 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). You can use the `SleepMode.cmd` script included in the RTC6 Software Package (under `RTC6 Tools`) to do this.
- See also [Section "Notes", page 36](#).

Notice!

- On some PCs it may happen that an RTC6 PCIe Board is not recognized by the device manager in a certain PCIe slot. Use a different slot.

5.4 Installing the RTC6 Software

Additionally to installing the RTC6 board driver, the RTC6 software must be "installed", that is, copied or unzipped on the hard drive of the PC.

RTC6 software for RTC6 PCIe Boards are the following files, see also [Section "Folder RTC6 Files", page 32](#):

- **RTC6 DLL**
 - `RTC6DLL.dll`
Win32-based RTC6 dynamic link library
 - `RTC6DLLx64.dll`
Win64-based RTC6 dynamic link library
- **RTC6 files**
 - `RTC6OUT.out`,
Program file for the PCIe-DSP
 - `RTC6ETH.out`
Program file for the Eth-DSP
 - `RTC6RBF.rbf`
Firmware file for the FPGA
 - `RTC6DAT.dat`
Binary auxiliary file

The files are included in the RTC6 Software Package. For easy identifying and archiving of different software versions, some of the files are also delivered zipped (the zip file names `RTC6<...>_<Version>.zip` include the version numbers).

To install the RTC6 software, follow these steps:

- Copy `RTC6DLL.dll` (or `RTC6DLLx64.dll`), `RTC6OUT.out` (with RTC6 Ethernet Boards `RTC6ETH.out`), `RTC6RBF.rbf` and `RTC6DAT.dat` (of the desired version) to your PC. Note that differing file versions cannot be arbitrarily combined with another (each zip file includes a text file with corresponding version information).

Notes

- Also provide the necessary correction file(s) (existing *.ct5 correction files can still be used; do not overwrite customized correction files!).

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 RTC6 PCIe Board and start up the control software (initialization of the board).
- (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 power supply for the scan system.
- (3) Turn off the peripheral equipment.
- (4) Close the control software and turn off the PC containing the RTC6 PCIe Board.



Caution!

- While the PC is turned on and off, short-term level variations at the output ports of the RTC6 PCIe Board, 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



Caution!

- 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 "Folder HPGL", page 31](#). This program lets you load graphics files in Hewlett Packard HPGL format (vector graphic plotter files `*.plt`) for transfer to the RTC6 PCIe Board.

- (1) Copy `Hpgl.exe` and the supplied `*.plt` files to the same folder as the `RTC6DLL.dll`, the [RTC6 files](#) and correction file(s).
- (2) Start `Hpgl.exe`.
- (3) Choose **Options > Correction**, and then select a correction file.
- (4) Choose **File > Load HPGL-File**, and then select a `*.plt`.
- (5) To start output, choose **Mark > Start Marking**.



5.7 User Programs and Demo Programs

The DLLs for RTC6 user programs ([RTC6DLL.dll](#), [RTC6DLLx64.dll](#)) support the RTC6 PCIe Board under 32-bit and 64-bit Microsoft operating systems Windows 10, 8, 7. The DLLs provide all necessary functions for operating the RTC6 PCIe Board.

Programming of user programs is described in detail in [Chapter 6 "Developing RTC6-User Programs", page 92](#). [Chapter 6.2 "Initialization and Program Start-Up", page 94](#) shows how to import the functions of the [RTC6 DLL](#) into user programs, if they are written in Pascal, C, C++ or C#.

On a 64-bit operating system, the 64-bit variant of the RTC6 board driver supports function calls from Win64 user programs as well as from Win32 user programs.

Therefore, existing Win32 user programs for the RTC6 PCIe Board are able to execute even on 64-bit systems, if the included Win32-based file [RTC6DLL.dll](#) is used.

For Win64 user program, the Win64-based file [RTC6DLLx64.dll](#) is included in the software package. In case a user program utilizes implicit linking to the [RTC6DLLx64.dll](#), it must be linked with the Win64-based import library [RTC6DLLx64.lib](#).

To help software developers get started, some example codes are shown in:

- [Chapter 6.2.5 "Example Code \(C\)", page 99](#)
- [Chapter 6.8.3 "Example Code \(C\)", page 132](#)
- [Chapter 7.1.4 "Example Code \(C\)", page 141](#)

Notice!

- 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.

6 Developing RTC6-User Programs

6.1 RTC6 Software Concept Basics

6.1.1 Controlling Scan Systems and Lasers – An Introductory Example

The SCANLAB RTC6 PCIe Board and its related software are designed for controlling scan systems and lasers.

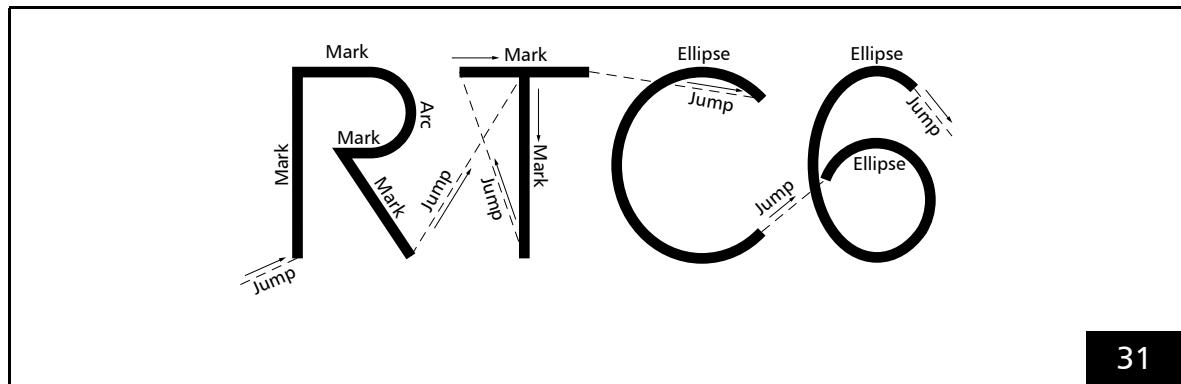
To illustrate the principle of operation, [Figure 31](#) shows a simple laser marking sample⁽¹⁾. The lettering is made up of straight line segments (vectors) and arc segments. The [RTC6 DLL](#) provides a set of jump, mark, arc and ellipse commands for laser processing along such segments. Each of these commands describes one vector or arc. Additional commands are available for controlling the laser during the marking process.

The RTC6 PCIe Board processes the commands it receives and precisely transmits the required marking signals to the scan head (using a predefined $10\ \mu\text{s}$ time raster) and to the laser. The scan head's galvanometer scanners accurately position their deflection mirrors in synchronization with the incoming control signals.

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

The current status of a scan head can also be queried via the RTC6 PCIe Board using appropriate commands.

For laser control, the RTC6 PCIe Board provides various analog and digital signal output ports freely available.



A laser marking sample.

6.1.2 Control Commands and List Commands

The RTC6 command set consists of:

- Control commands
- 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 RTC6 PCIe Board, 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 104](#).

List commands sent to the RTC6 PCIe Board afterwards are not executed immediately. They are stored in a list memory first. The commands from the list memory are only processed in real time after the list has been started.

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 mark speed.

Some of the list commands are so called *short list commands*. These do not require the full 10 µs clock cycle for command execution. They are executed along with the next list command, one directly after the other within a single 10 µs clock cycle. A control command cannot be executed in between. The total list execution time is thereby reduced.

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

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 RTC6 commands are described in detail in [Chapter 10 "RTC6 Commands"](#).

An overview is provided in [Chapter 10.1 "Overview", page 301](#).

6.2 Initialization and Program Start-Up

To use the commands and functions of the RTC6 PCIe Board in a user program:

- A complete installation must have been carried out – this includes the RTC6 hardware, RTC6 board driver and RTC6 software, see [Chapter 5 "Installation and Start-Up", page 87](#)
- The desired **RTC6 DLL** (Win32- or Win64-based) must be linked to the user program, see [Chapter 6.2.1 "DLL Calling Convention", page 94](#)
- The **RTC6 DLL** functions must be imported to the user program, see [Chapter 6.2.2 "Importing Commands", page 94](#)

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

- that initialize the **RTC6 DLL** for the calling user program and assigns access rights for the installed RTC6 PCIe Boards, see [Chapter 6.2.3 "Initializing the RTC6 DLL and RTC6 Board Management", page 96](#)
- that initialize the installed RTC6 PCIe Boards, see [Chapter 6.2.4 "Start of RTC6 PCIe Board Operation", page 97](#)

Only afterward can the user program load the desired command lists into the RTC6 list memory and start processing.

These steps are described individually in the following chapters. They are shown by an example program in [Chapter 6.2.5 "Example Code \(C\)", page 99](#).

6.2.1 DLL Calling Convention

Link the user program to the **RTC6DLL.dll**/
RTC6DLLx64.dll⁽¹⁾.

The **RTC6 DLL** calling convention is `stdcall` for Win32 and `fastcall` for Win64.

The structure alignment is 4 byte for Win32 and 8 byte for Win64.

6.2.2 Importing Commands

To facilitate importing the commands of the **RTC6 DLL** into a C, C++, C# or Pascal user program, the RTC6 Software Package contains corresponding utility files.

Notice!

- Some RTC6 commands can only be *completely* executed with appropriate options, see also [Chapter 2.6 "Options", page 37](#).
- **get_RTC_version** provides information about the options installed on your RTC6 PCIe Board.

(1) See [Chapter 5.4 "Installing the RTC6 Software", page 89](#).

Pascal

Use the file `RTC6Import.pas` as a unit and call the RTC6 commands you need, like

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

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 `RTC6impl.h` and link with the (C) import library `RTC6DLL.lib` (for Win32 user programs or with `RTC6DLLx64.lib` for Win64 user programs) for building the executable.

Call the RTC6 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 `RTC6expl.h`. Before calling any RTC6 command, initialize the **RTC6 DLL** by calling the function `RTC6open` (which is defined in the file `RTC6expl.c`). When you have finished the RTC6 session, close the **RTC6 DLL** by calling the function `RTC6close` (also defined in the file `RTC6expl.c`).

For building the executable, link with the file `RTC6expl.obj`, which you can generate from the `RTC6expl.c` source code.

Call the RTC6 commands you need, like

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

Differences between Implicit and Explicit Linking

	Implicit Linking	Explicit Linking
Necessary Files	<code>RTC6impl.h</code> or <code>RTC6impl.hpp</code> , <code>RTC6DLL.lib</code> or <code>RTC6DLLx64.lib</code>	<code>RTC6expl.h</code> , <code>RTC6expl.c</code>
Advantages	Easiest linking method.	Eliminates the need to link the user program with an import library.
Disadvantages	Users need to link the user program to a compiler-specific import library.	Users need to initialize (<code>RTC6open</code>) and close (<code>RTC6close</code>) the RTC6 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 `RTC6impl.hpp` instead of `RTC6impl.h`. Otherwise, the command import is realized as in C (see above).

C#

For implicit linking of the **RTC6 DLL** in C# the wrapper class is provided. It also supports the 'Any CPU' option for simultaneous usage with 32-bit and 64-bit programs.

6.2.3 Initializing the RTC6 DLL and RTC6 Board Management

As many RTC6 PCIe Boards as the PCIe bus allows can be operated simultaneously in one PC. However, the RTC6 board management can manage a maximum of 255 RTC6 PCIe Boards and RTC6 Ethernet Boards at the same time, see [Chapter 16.5.3 "About the RTC6 Board Management", page 887](#).

The **RTC6 DLL** allows multi-processing⁽¹⁾ as well as multi-threading.

No RTC6 PCIe Board can be simultaneously used by several user programs. Access rights (even if temporary) to existing boards are assigned on an exclusive basis by the **RTC6 DLL**.

Multiple threads of *one* user program can use the same board, but can not send commands to it at the same time (the **RTC6 DLL** automatically serializes the command calls).

The RTC6 board management, see [Chapter 16.5.3 "About the RTC6 Board Management", page 887](#), must be initiated by **init_rtc6_dll** at the beginning of each user program so that an RTC6 PCIe Board can be addressed at all. This even applies, if only *one* RTC6 PCIe Board is to be used by *only one* user program.

init_rtc6_dll does the following (for details, see the command description):

- Searches for all present RTC6 PCIe Boards
- Establishes the RTC6 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 **RTC6 DLL**-internal numbers for all found RTC6 Ethernet Boards (important for multi-board commands)
- Sets one board as the *active* board, which therefore, is the target for non-multi-board commands
- Does *not* search for RTC6 Ethernet Boards. These must be added separately, see [Chapter 16 "Appendix A: The RTC6 Ethernet Board", page 858](#).

(1) Several user programs can run simultaneously.

acquire_rtc, **release_rtc**, **select_rtc** allow changing access rights and the active board at run-time.

Usage of several boards is described in [Chapter 6.6 "Using Several RTC6 PCIe Boards in One PC", page 122](#); usage by several user programs is described in [Chapter 6.7 "Usage of RTC6 PCI Express Boards by Several User Programs", page 127](#).

After RTC6 DLL initialization by **init_rtc6_dll**, the default DLL-operation mode is the **RTC6 Standard Mode**. If required, a different DLL-operation mode can be set (see **set_rtc4_mode**, **set_rtc5_mode** and **set_rtc6_mode**).

The **RTC4 Compatibility Mode** is provided so that user program written for the RTC4 can be processed by the RTC6 PCIe Board (for otherwise unchanged commands⁽²⁾) without modification. In the command descriptions, notes changes in this regard, see [Chapter 10.2 "RTC6 Command Set", page 314](#), "RTC4→RTC6" rows.

See also [Chapter 2.10.2 "Porting RTC4 Source Code to the RTC6 PCIe Board", page 48](#).

The similar applies to the **RTC5 Compatibility Mode**, see RTC5→RTC6 rows of the command descriptions. See also [Chapter 2.12.2 "Adapting RTC5 Source Code for the RTC6 PCIe Board", page 59](#).

(2) Commands having identical names with analogous parameterization and meaning.

Example: **mark_abs**(X, Y).

6.2.4 Start of RTC6 PCIe Board Operation

At the beginning of every RTC6 user program – after initialization of the [RTC6 DLL](#), see [Chapter 6.2.3](#)

["Initializing the RTC6 DLL and RTC6 Board Management", page 96](#) – it is recommended to carried out the following sequence of steps in order to start RTC6 PCIe Board operation.

- (1) [Initializing the Board, page 97](#)
- (2) [Configuring the Board, page 97](#)
- (3) [Initializing the Scan System Control, page 98](#)
- (4) [Initializing the Laser Control, page 98](#)
- (5) [Loading and Executing Lists, page 98](#)

Initializing the Board

- Call [load_program_file](#). For the individual actions, see [Function, page 500](#).

After execution of [load_program_file](#), the position output is automatically set to the null position (0|0)⁽¹⁾ and laser control is deactivated⁽²⁾⁽³⁾.

In order to be able to combine [RTC6 files](#)

([RTC6DLL.dll/RTC6DLLx64.dll](#), [RTC6OUT.out/RTC6ETH.out](#), [RTC6RBF.rbf](#), [RTC6DAT.dat](#)), they must have certain file versions, see also [Chapter 5.4 "Installing the RTC6 Software", page 89](#). [load_program_file](#) performs a version compatibility check. If a version error exists (error code [7](#) and [get_last_error](#) return code [RTC6_VERSION_MISMATCH](#)), the board is released by [release_RTC](#). Thus, it is not available for further RTC6 commands other than those not requiring granted access rights.

RTC6 PCIe Boards can only be operated if all [RTC6 files](#) are available with a compatible combination of versions, see [Chapter 5.4 "Installing the RTC6 Software", page 89](#). If the version check fails and the board is not acquired by another user program, then [load_program_file](#) can also be used at any time to load a correct program version. After that, the board can be acquired by [select_RTC](#) or [acquire_RTC](#). If several RTC6 PCIe Boards are master/slave connected, see [Chapter 6.6.3 "Master/Slave Operation", page 123](#).

Configuring the Board

- If necessary, configure the RTC6 list memory by [config_list](#). After [load_program_file](#) the list memory areas "List 1" and "List 2" can each store 4,194,304 list commands. The protected list memory area "List 3" cannot store list commands, see [Figure 32](#).

(1) Center of the [Image](#) field.

(2) There are no laser control signals at the corresponding pins (LASERON, LASER1, LASER2). They are in high-impedance state.

(3) On the state of the various output ports, see [page 501](#).

Initializing the Scan System Control

(1) Use `load_correction_file` to download the necessary correction file(s) to the RTC6 PCIe Board (you can load a correction table before or after `load_program_file` but you should load it *before* `select_cor_table`; you should at least load a 1:1 correction table).

See [Chapter 8.5 "Controlling 2D Scan Systems and 3D Scan Systems", page 235](#), for information about using several different correction tables.

(2) Assign the previously loaded correction table(s) to the scan head control port(s) by `select_cor_table`⁽¹⁾. This causes the intended **Image field** correction to also be applied to the default position (0|0) previously set by `load_program_file` (in some circumstances, **Image field** correction can even shift the null position).

After `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 command** or **Mark command**.

(3) Define the scanner delay mode by `set_delay_mode` (among others, "Variable Polygon Delay" or constant Polygon Delay).

(4) If necessary⁽²⁾, load a table for the "Variable Polygon Delay" by `load_varpolydelay`.

The remaining settings (**Scanner Delays**, jump speed and mark speed) are set by further control commands or list commands.

(1) `load_correction_file` automatically calls `select_cor_table` with the last assigned table numbers after loading the correction table (if `load_correction_file` follows after `load_program_file`), see [Section "Notes", page 175](#).

(2) Step 4 can be omitted, if a new `RTC6DAT.dat` is created by `create_dat_file` after loading the table. Then, this table is automatically loaded with the next `load_program_file`.

Initializing the Laser Control

(1) Set the laser mode by `set_laser_mode`.

(2) Set the polarity of the laser control signals appropriate to your laser system by `set_laser_control`.

(3) Set the FirstPulseKiller length (only for YAG modes) by `set_firstpulse_killer`.

(4) Set the delay of the Q-Switch signal (only for YAG modes, in particular for YAG Mode 5) by `set_qswitch_delay`.

(5) Set the stand-by pulses by `set_standby` (in particular for **Laser Mode 4** and **Laser Mode 6**).

(6) Enable the laser control signals (see `enable_laser`), if they have been suppressed by `set_laser_control`.

The remaining settings (such as laser timing or **Laser Delays**) are set by further control commands or list commands, for example, see [Chapter 7.1.4 "Example Code \(C\)", page 141](#) or [Section "Signals for "Laser Active" Operation", page 185](#).

Loading and Executing Lists

(1) Load the list(s).

(2) If necessary, enable the external start input by `set_control_mode`.

Notice!

- 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 (C)

The following C source code for a console user program (environment: Win32) illustrates the programming fundamentals of **RTC6 DLL** and **RTC6 PCIe Board** initialization (for complete demo programs, see [Chapter 11 "Demo Programs", page 848](#)).

Necessary sources: **RTC6impl.h**, **RTC6DLL.lib** (for implicit linking) or **RTC6expl.h**, **RTC6expl.c** (for explicit linking) to link the **RTC6 DLL** to the program, see [Chapter 6.2.1 "DLL Calling Convention", page 94](#). If the operating system does not find the **RTC6DLL.dll** on user program startup, it produces a corresponding error message and terminates the program.

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

// RTC6 header file for implicitly linking to the RTC6DLL.dll (for building the executable, also link
// with the (Visual C++) import library RTC6DLL.lib):
#include "RTC6impl.h"
// Alternatively: RTC6 header file for explicitly linking to the RTC6DLL.dll
// (for building the executable, link with the file RTC6expl.obj, which you can generate
// from the RTC6expl.c source code):
//#include "RTC6expl.h"

void __cdecl main( void*, void* )
{

    // only for explicitly linking:
    // if ( RTC6open() ) // error detected, RTC6open returns 0 for no error
    // {
    // printf( "Error: RTC6DLL.dll not found\n" );
    // terminateDLL();
    // return;
    // }

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

    UINT ErrorCode;

    // Initializing the RTC6 DLL (the following command must be called as the first RTC6 command)
    ErrorCode = init_rtc6_dll();

    // Following init_rtc6_dll you should include a program code to catch an error during
    // initialization, for example, for the case the desired board is not detected, access is denied
    // or for another error (for example, a version mismatch).
    // See Chapter 6.8.3 "Example Code \(C\)", page 132.
}
```



```
// Initializing the RTC6 PCIe Board:  
// - Selecting the board number 1 as the active board in this user program.  
// - If desired: Selecting the RTC4 Compatibility Mode as operation mode  
// (default: RTC6 Standard Mode)  
// - Optional: stopping any list running on RTC6 PCIe Board number 1 (if it has been used  
// previously by another user program, a list might still be running). load_program_file does this  
// automatically itself. Otherwise, load_correction_file cannot be executed before  
// load_program_file.  
// - Calling load_program_file for initializing the board, loading the program file, etc.  
// (here also a program code should be included to catch possible errors - for example, file or  
// system errors - during initialization, see Chapter 6.8.3 "Example Code (C)", page 132).  
// - Clearing all previous error codes  
// - (stop_execution might have created an RTC6_TIMEOUT or RTC6_BUSY error).  
// - Configuring the RTC6 list memory, default: 4,194,304 for list 1 and list 2 each.  
void) select_rtc( 1 );  
set_rtc4_mode();  
  
// Optional: stop_execution();  
  
ErrorCode = load_program_file( 0 ); // Path = 0: path of current working directory  
if ( ErrorCode )  
{  
    printf( "Program file loading error: %d\n", ErrorCode );  
    free_rtc6_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 in Chapter 7.1.4 "Example Code (C)", page 141.  
  
// End of main program  
terminateDLL();  
return;  
}  
  
void terminateDLL()  
{  
    printf( "- Press any key to shut down \n" );  
    while( !kbhit() );  
    (void) getch();  
    printf( "\n" );  
    // Close the RTC6 DLL  
    free_rtc6_dll();  
    // only for explicitly linking:  
    // RTC6close();  
}
```

6.3 RTC6 List Memory

The RTC6 list memory serves as intermediate storage for list commands.

Before list commands can be transferred to the RTC6 PCIe Board, 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 104](#).

By additional control commands, the processing of the transferred list commands can be started.

6.3.1 Lists and the Protected List Memory Area

The RTC6 list memory offers $8M = 8,388,608 = 2^{23}$ storage positions in total.

In general, it can be split into three areas. Their sizes are freely configurable.

- Two list memory areas, "List 1" and "List 2". In this manual, these are also simply designated as "lists".
- User-definable is in addition: a third "protected list memory area", "List 3". This is protected against unintended overwriting (by loading normal command lists).

"List 1" and "List 2"

In principle, both list memory areas "List 1" and "List 2" can be used in a manner identical to the two list memory areas of the RTC5 or RTC4 boards, for example, for continuous loading and processing of command lists.

However, the RTC6 PCIe Board has a bigger total list memory and furthermore, the size of each memory area can be freely configured, see [Chapter 6.3.2 "Configuring the RTC6 List Memory", page 102](#).

For list handling, see [Chapter 6.4 "List Handling", page 104](#).

"List 3" – Protected List Memory Area

"List 3" is intended for a protected storage of frequently needed list command sequences (as subroutines or character set definitions). It is protected against unintended overwriting (during loading of normal command lists).

There are principally two alternative ways to utilize this protection feature:

- (1) Subroutines can be written to the upper positions of the list area. These subroutines can be subsequently assigned to the protected list memory area "List 3". Such subroutines are called – both initially in the list memory area as well as subsequently in the protected list memory area "List 3" – by `list_call`, specifying an absolute memory position.
- (2) Special commands allow subroutines and character set definitions to be loaded directly in the protected list memory area "List 3" as indexed subroutines or definitions. They can then be called by providing the corresponding index. Indexed character set definitions can, for example, be used in conjunction with `mark_text` for directly marking text.

SCANLAB strongly recommends *not* intermixing usage of these two methods. Otherwise, unintended data loss by overwriting can occur even in the protected list memory area "List 3".

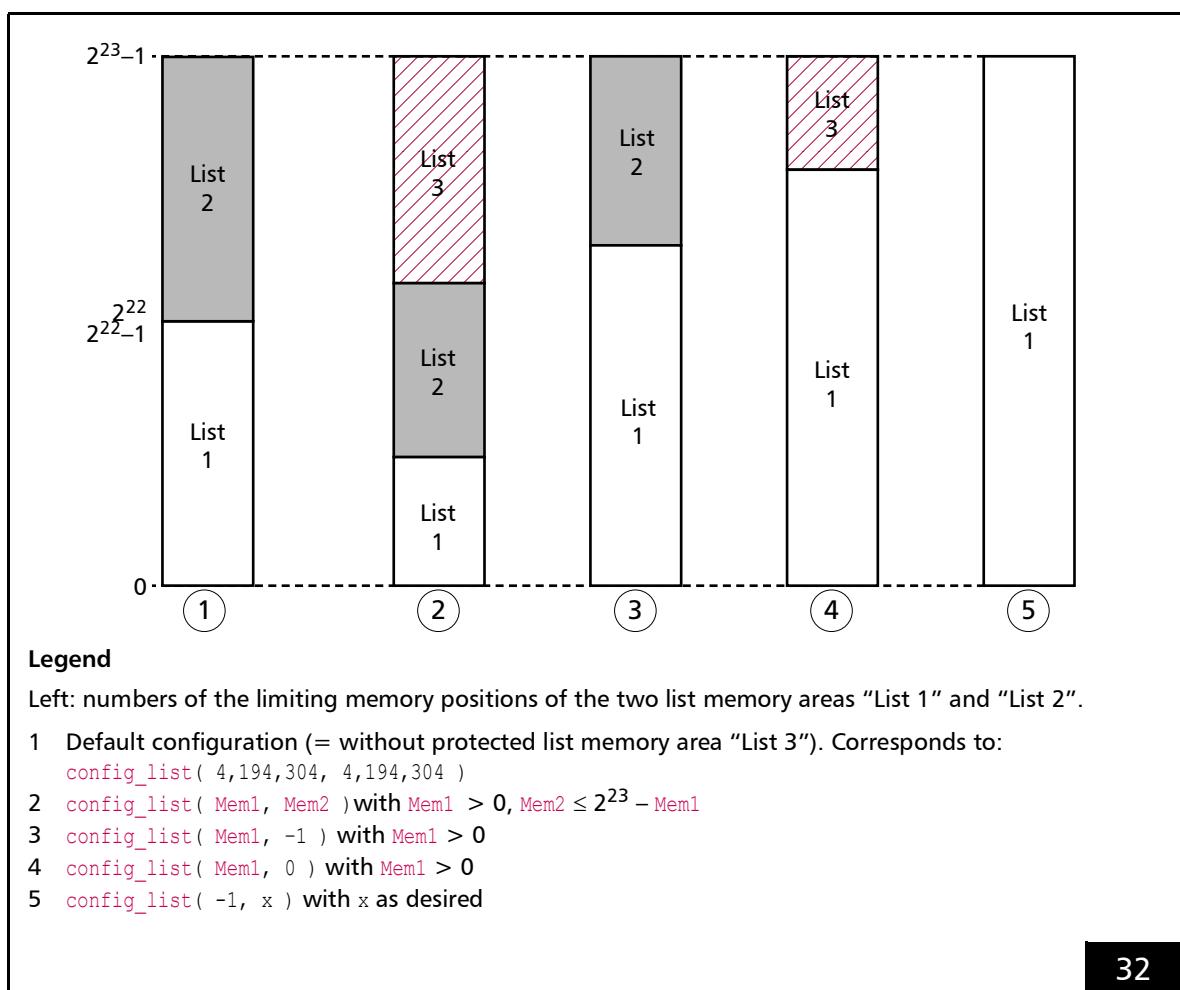
The RTC6 command set includes appropriate conversion commands so that users are not forced to continuously use only one of the two methods. The defining of subroutines and character sets, as well as their management and subsequent conversion options are detailed in [Chapter 6.5 "Structured Programming", page 111](#).

6.3.2 Configuring the RTC6 List Memory

The RTC6 list memory offers $8M = 8,388,608 = 2^{23}$ storage positions in total. After `load_program_file` the list memory areas "List 1" and "List 2" can each store 4,194,304 list commands. The protected list memory area "List 3" cannot store list commands, see [Figure 32](#).

By `config_list`, the list memory areas can be reconfigured.

If a user program only needs *one single* list, then the total RTC6 list memory can be treated as a single list memory with a total capacity of 2^{23} storage positions, see Configuration (5) in [Figure 32](#).



Examples of allowed list memory configurations.



Generally, during configuration are assigned:

- lower storage position numbers to "List 1"
- medium storage positions numbers to "List 2"
- upper storage position numbers to "List 3"

When configuring the list memory areas, the following must be observed:

- "List 1" must contain at least one storage position
- the total sum of storage positions for "List 1" and "List 2" must not exceed 2^{23} .

Other than that, the sizes of "List 1" and "List 2" can be set as desired. For example, "List 2" can be configured even with 0 storage positions, see configuration (4) and (5) in [Figure 32](#).

With every configuration, remaining storage positions (not assigned to "List 1" or "List 2") are automatically assigned the protected list memory area "List 3".

The memory content is not altered by the configuration process. Therefore, repeating the call with differing parameters is nondestructive.

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

- List boundaries should not be moved to within an eventual subroutine.
- The protection of a "List 3" range is removed, if this range is assigned to list memory area "List 1" or "List 2".
- Valid jump addresses specified in [Jump commands](#) might become invalid if the configuration is altered, see [Chapter 6.5.3 "Jumps", page 119](#).
- After the protected list memory area "List 3" has been made larger, defragmentation might be needed to make the newly assigned memory area usable, see [Section "Index Management and Defragmentation", page 114](#).

6.4 List Handling

The two list memory areas "List 1" and "List 2" serve as intermediate storage for the continuous loading and processing of list commands.

6.4.1 Loading Lists

"List 1" and "List 2" are enabled to be filled with list commands by `set_start_list_pos`, `load_list` or other control commands (see below). 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. When in the process of list loading the list end 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 list memory area "List 3".

In general, when list commands are loaded into storage positions, any list commands previously stored there are overwritten. This occurs even if they have not yet been processed or are currently being executed. User should make sure not to overwrite commands still needed by the user program (see below).

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

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 called, this automatically results in a flush. Thereby, the still buffered list commands are transferred to the list memory.

A flush can be initiated at any time by `set_input_pointer(get_input_pointer())`, even if the buffer is yet "incomplete". This is only necessary in some circumstances when list commands should be processed and list input is not yet finished (for example, with an `External Start`).

"Unconditional" Loading

The input pointer is set:

- to the beginning of the selected list by
 - `set_start_list`
 - `set_start_list_1`
 - `set_start_list_2`
- to the specified address of the selected list by
 - `set_start_list_pos`
 - `set_input_pointer`

The next list command is written to this address regardless of the current status of the specified list, see [Chapter 6.4.2 "List Status", page 106](#).

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



Loading with Protection

The loading process is initialized by **load_list**, which sets the input pointer to the specified address in the selected list (just like **set_start_list_pos**). However, this occurs 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 RTC6 PCIe Board automatically determines the corresponding appropriate list).

The return value of **load_list** reveals if, and in which list, the loading procedure has been successfully initialized.

Otherwise, the input pointer is set to an invalid position. Then, no further list commands can be input until the input pointer is correctly set back to a valid position again (for example, by repeating **load_list** with a positive result or **set_start_list_pos**).

This automatically prevents unintentional overwriting of commands that are still to be executed.

load_list is useful in scenarios such as alternating list changes, where you want to wait specifically for a list to be processed, see [Section "Alternating List Changes", page 110](#).

Terminating Lists

A command list can be, but need not necessarily be, terminated by a **set_end_of_list**.

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 has been terminated by **set_end_of_list**, see [Chapter 6.4.6 "Changing Lists Automatically", page 109](#).

The loading of a **set_end_of_list** does *not* stop the loading procedure itself. Therefore, list commands immediately following a **set_end_of_list** are still loaded into the same list.

6.4.2 List Status

Dependent on the command input and output statuses, lists receive particular list status values (compare to [Chapter 6.4.3 "List Execution Status", page 107](#)).

By control command **read_status**, the current list status values can be queried – separately for both lists.

- **LOAD** list status

The **LOAD** list status **LOAD1** or **LOAD2** indicates that the input pointer is currently in this list. In any case, the **LOAD** list status of the other list is *not* set.

- **READY** list status

The **READY** list status **READY1** or **READY2** is set when a **set_end_of_list** is written into the list during the loading procedure. The **READY** list status is reset when the **LOAD** list status of the list is newly set.

- **BUSY** list status

The **BUSY** list status **BUSY1** or **BUSY2** indicates that the output pointer is currently in this list after list execution (of "List 1" or "List 2") has been started. In any case, the **BUSY** list status of the other list are then *not* set. The **BUSY** list status of a list is reset when **set_end_of_list** is executed (or is alternating set, if automatic list changing has been previously activated). If a list is opened for loading while still being processed, its **BUSY** list status still remains set.

- **USED** list status

The **USED** list status **USED1** or **USED2** is set when a **set_end_of_list** is reached during processing. The **USED** list status is reset when the **LOAD** list status of the list is set.

Notes

- If the list status is queried during processing of a subroutine in the protected list memory 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 has been 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 Stop**), the **USED** list status is set for both lists (as by initialization). See also **load_list**(*ListNo* = 3).
- If you want to explicitly set the **USED** list status for a list *ListNo* (for example, after abortion by **stop_execution** or an **External Stop**), then load a **set_end_of_list** 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 **USED** list status afterward.
- When interpreting the status values read back 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

In addition to the list status values, see [Chapter 6.4.2 "List Status", page 106](#), list execution status values are provided.

These can be queried by the control command `get_status`.

- **BUSY** list execution status

The **BUSY** list execution status is set when:

- the RTC6 PCIe Board currently processes one of the two lists
- a list has been paused by the control commands `pause_list` or `stop_list`

The **BUSY** list execution status is not set when a list has been paused by the list command `set_wait` (is set again by a subsequent `release_wait`).

- **PAUSED** list execution status

The **PAUSED** list execution status is set when processing of the list has been paused by `pause_list`, `stop_list` or `set_wait`. It is reset by a subsequent `restart_list` or `release_wait`, see also [Chapter 6.4.5 "Interrupting Lists for Synchronization of Processing", page 109](#).

- **INTERNAL-BUSY** list execution status

The **INTERNAL-BUSY** list execution status is set when the RTC6 PCIe Board is busy with executing a control command, which needs more than $10 \mu\text{s}$ for executing a scan movement (for example, `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`).

Notes

- The **INTERNAL-BUSY** list execution status and the **BUSY** list execution status cannot be set at the same time.
- With the RTC6 PCIe Board, the **BUSY** list execution status is also available as **BUSY OUT** signal at:
 - **LASER Connector**, see [Figure 17](#)
 - **EXTENSION 1** socket connector, see [Figure 22](#)
 - **MARKING ON THE FLY** socket connector, see [Figure 25](#)
- Some control commands are ignored (not executed), when the **BUSY** list execution status and/or **INTERNAL-BUSY** list execution status are set (for example, `auto_cal`, `goto_xy`, `load_correction_file`) or – with set **INTERNAL-BUSY** list execution status – are only executed with delay after the **INTERNAL-BUSY** list execution status has been reset again (for example, `execute_list_pos`, `set_offset`).

6.4.4 Starting and Stopping Lists

List processing ("List 1" or "List 2") can be started by:

- The control command `execute_list`
- An external start signal, see [Chapter 9.3.1 "Starting and Stopping Lists by External Control Signals and Master/Slave Synchronization", page 288](#)

`execute_at_pointer` can be used to start output of a list at a specified address. If an external start signal is used, see [Chapter 9.3.1 "Starting and Stopping Lists by External Control Signals and Master/Slave Synchronization", page 288](#), then

`set_extstartpos_list` allows definition of a start address for the [External Start](#).

The RTC6 PCIe Board starts the execution immediately. Even during 10 μ s-clocked execution of the list commands, you can still send control commands to the RTC6 PCIe Board. These are immediately executed 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). However, the second list can only be started after processing of the first list has been finished. During list processing, `execute_list` or an external start signal is ignored.

Execution of a list can also be stopped at any time, for example, for implementing an emergency shutdown. As soon `stop_execution` is called or an external stop signal is transferred to the RTC6 PCIe Board, the currently executed list is aborted and the [Signals for "Laser Active" Operation](#) are turned off (but not deactivated).

With `range_checking`, the processing of a list can also be terminated automatically (like with `stop_execution`).

If, during list processing, the list end is reached without encountering a `set_end_of_list`, then processing continues at the beginning of the current list. This is repeated until either `stop_execution` is called or an external stop signal is transferred to the RTC6 PCIe Board.

If during list processing a `set_end_of_list` is reached, then list execution stops - unless `auto_change`, `auto_change_pos` or `start_loop` has been previously called, a list change takes place, see [Chapter 6.4.6 "Changing Lists Automatically", page 109](#). This list change occurs only upon reaching a `set_end_of_list`.

Notes

- Lists are not automatically started. Regardless of how many commands are loaded, a list must be started as described in order to be processed.
- To also enable starting and stopping of list execution by external signals, the RTC6 PCIe Board provides corresponding control input ports, see [Chapter 9.3.1 "Starting and Stopping Lists by External Control Signals and Master/Slave Synchronization", page 288](#).
- `set_pause_list_cond` or `set_pause_list_not_cond` can be used to set a condition for the 16-bit input port so that a `pause_list` is executed instead of `stop_execution` when an [External Stop](#) is present. In The list execution can then be continued by `restart_list`.

6.4.5 Interrupting Lists for Synchronization of Processing

The list command `set_wait` makes it possible to insert numbered break points ("wait markers") into a list. Each break point is associated with a number greater than zero. When the RTC6 PCIe Board reaches a break point during list execution, see [Chapter 6.5 "Structured Programming", page 111](#), output of the list is temporarily interrupted and the laser is switched off.

`get_wait_status` checks whether list processing is currently interrupted at a break point. If processing is interrupted, `get_wait_status` returns the number (wait_word) of the break point (otherwise the value zero).

Break points are provided for synchronization purposes. The user program should perform a handling routine for each break point. When that handling routine is finished, list processing can be resumed (at the list command that follows) by the control command `release_wait`.

By `set_wait` the **PAUSED** list execution status (queryable by `get_status`) is set and the **BUSY** list execution status is reset. The opposite occurs after a subsequent `release_wait`.

List execution can be interrupted at any desired point in time by the control command `pause_list` (or by the synonym `stop_list`) and resumed by `restart_list`.

By `pause_list`, [Signals for "Laser Active" Operation](#) are suppressed and the scan system remains in the most recently defined state – even if in the middle of [Microstepping](#). After a subsequent `restart_list`, the scan system resumes the planned movements (of the current command) and the laser control signals are released again (in general, an interrupted marking cannot be continued without a disruption in the marking result).

By `pause_list` the **PAUSED** list execution status (queryable by `get_status`) is set and is reset by `restart_list`. The **BUSY** list execution status is left unchanged by both commands.

6.4.6 Changing Lists Automatically

If the RTC6 list memory is configured for two list memory areas "List 1" and "List 2", see [Chapter 6.3.2 "Configuring the RTC6 List Memory", page 102](#), then a second list can be loaded while the first list is still being processed.

It typically takes substantially longer to process a list than to write it into the memory. Continuous processing of arbitrarily long lists is therefore also possible, if they are divided into command blocks.

Continuous command output, which requires switching between two lists, can be achieved by automatic list changing as described in the following sections.

The commands for automatic list changing only take effect when the next following `set_end_of_list` is executed. That is, automatic list changing after processing a list can only occur if that list has been finished with a `set_end_of_list`. Otherwise, processing resumes at the beginning of the same list.

If the RTC6 list memory is configured for a list memory area ("List 2") to size 0, then all automatic list change commands lead to "List 1" to the specified position.

One-Time List Change

`auto_change` and `auto_change_pos` activate an automatic, one-time list change between "List 1" and "List 2". After processing of the current list (when `set_end_of_list` is reached), processing of the next list is thereby automatically started.

When using `auto_change` the next list is started at position 0; when using `auto_change_pos` the next list is started at the specified start position (list memory address as an offset to the beginning of the list).

Alternating List Changes

Another way to achieve continuous command output is by alternatingly repeating output of the two lists.

To do so, **start_loop** must be called. This causes a continuous, automatic and alternatingly repeating processing of both lists, provided both lists are finished each with **set_end_of_list**.

The alternating processing repeats until **quit_loop** is called. **quit_loop** terminates continuous processing as soon as the current list is finished.

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

Notes

- The commands for starting a one-time automatic list change and **start_loop** to start an alternating list change can be called at any point in time. However, they do not take effect until the next **set_end_of_list** is reached.
- When loading a list while another is being processed, make sure no still-needed commands are thereby overwritten. Useful here is **load_list**, which only starts loading a list if it is currently not in use or already has been processed, see [Chapter 6.4.1 "Loading Lists", page 104](#).
- Moreover, the currently new list should have made a certain amount of loading progress before the list change occurs. The input pointer should always be adequately ahead of the output pointer (because the PCI transfer of the list commands is buffered, see [Chapter 6.4.1 "Loading Lists", page 104](#), and so-called short list commands can be used, see [Chapter 6.1.2 "Control Commands and List Commands", page 93](#)). Otherwise, "old" commands might be unintentionally executed.
- The RTC6 PCIe Board does not support the RTC4 circular queue mode, see [Chapter 6.5.4 "RTC4-Circular Queue Mode", page 120](#). However, this operating mode can also be effectively replaced using an alternating list change and **load_list** described above.

6.5 Structured Programming

The RTC6 command set supports structured programming and output of list commands by numerous ways to define subroutines and character sets, as well as list commands for controlling program flow.

6.5.1 Subroutines

- As list-command sequences, subroutines can principally be located in any part of the list memory.
- Preferably, subroutines should be written to a upper portion of the list memory, see [Section "List 3" – Protected List Memory Area", page 101](#).
- A list boundary should not run through a subroutine.
- A subroutine must be terminated with a [list_return](#).
- It can be defined:
 - [Non-Indexed Subroutines](#)
 - [Indexed Subroutines](#)

Non-Indexed Subroutines

As with "normal" list-command sequences, non-indexed subroutines are loaded into a list memory area ("List 1" or "List 2") by list-loading commands (see [Chapter 6.4.1 "Loading Lists", page 104](#)). Each subroutine must be terminated with a [list_return](#). It is called by [list_call](#) with a parameter specifying the absolute memory address.

After the subroutine (including the terminating [list_return](#) command) has been processed, it is continued with the command that follows the calling position.

Notes

- Non-indexed subroutines cannot be written directly to the protected list memory area "List 3". However, they can be subsequently protected, see also [Section "Subsequent Protection and Conversion of Non-Indexed Subroutines", page 115](#). For the subroutine to be indexed for this purpose with [set_sub_pointer](#), however, its start address must be known. Prior to loading a non-indexed subroutine into the list memory area "List 1" or "List 2", you should therefore always read out the start address by [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. Otherwise, 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. 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](#) but not the command which follows after [list_call](#) (which would process the subroutine once again having an uncorrelated [list_return](#)).

Indexed Subroutines

load_sub assigns a desired index to a subroutine (which is defined by subsequent list commands), and loads it into the protected list memory area "List 3".

An indexed subroutine must be terminated by a **list_return** (otherwise it is not stored). It is called by **list_call** (with a parameter specifying the index).

A maximum of 1024 subroutines can be stored.

The management of the indexed subroutines occurs on the RTC6 PCIe Board automatically.

For more information on index management, see [Section "Index Management and Defragmentation", page 114](#).

The memory address of an indexed subroutine can be queried by **get_sub_pointer**. With it, the indexed subroutine can be called by specifying the absolute memory address (just as with non-indexed subroutines).

An indexed subroutine is only stored by **load_sub** under the following circumstances:

- If prior to loading, configuration of "List 1" and "List 2" resulted in a protected list memory 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.
- **get_list_space**, if called after a **load_sub** (but before the terminating **list_return**), can be used for querying the amount of still-available memory in the protected list memory area "List 3"
- If the indexed subroutine is terminated by a **list_return**
- If **list_return** is preceded by no other command for positioning the input pointer (for example, another **load_sub**, **set_input_pointer**, or **set_start_list_pos**)
- If the index is within the valid range (0...1023)

After a **list_return**, the input pointer becomes invalid. Any subsequent list commands are no longer stored.

Indexed subroutines are written to "List 3" by **load_sub** commands in order of entry. The starting address is automatically set after the end of the last subprogram.

If an indexed subroutine is stored using an already-existing index, then the prior subroutine with that same index is not overwritten. It remains in the protected list memory area "List 3", though it is no longer indexed. Therefore, it can no longer be called through its index by **sub_call** (whereas it can be still called through its absolute memory address by **list_call**).

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

To load an indexed subroutine into the protected list memory area "List 3" that is already fully loaded with indexed subroutines, you must first appropriately expand the size of the protected list memory area "List 3" by **config_list** and then defragmenting it by **save_disk/load_disk**. Note that expanding the size alone is not sufficient, see [Section "Index Management and Defragmentation", page 114](#).

Notes on Not-Indexed Calls

- Index management or defragmentation, see [Section "Index Management and Defragmentation", page 114](#), can result in a change of the indexed subroutine absolute memory address. SCANLAB therefore advises against calling an indexed subroutine by **list_call**.

Rules for Programming

Observe the following guidelines when programming indexed subroutines:

- In an indexed subroutine, **set_end_of_list** is replaced by a **list_nop**.
- Absolute jumps within or out from the protected list memory area "List 3" are ignored during processing, see [Chapter 6.5.3 "Jumps", page 119](#). Therefore, absolute jumps cannot be used in indexed subroutines.
- When the subroutine is processed, also ignored are:
 - Relative jumps that exceed the boundaries of an indexed subroutine
 - **Jump commands** which initiate a jump to themselves

General Information on Calling Subroutines

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

When calling with **sub_call** or **list_call**, only relative **Jump commands** and **Mark commands** may be used, if the subroutine execution is to be repeated at different **Image field** places.

To be able to use the absolute **Jump commands** and **Mark commands**, which are often easier to handle, the so-called "**AbsCalls**" are provided.

"AbsCalls"

If the subroutines contain only relative **Vector commands** and "**Arc**" **commands**, see [Chapter 7.1.1 "Marking with Vector Commands and "Arc" Commands", page 135](#), the corresponding processes can be repeated at different places in the **Image field**.

With "**AbsCalls**" the current position is taken over as offset. This offset is then taken into account for all subsequent **Vector commands** 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 are made with **list_call_abs** and **sub_call_abs**.

Conditional Calls

To enable calling of subroutines dependent on external control signals, additional commands are available for conditional branching during program execution, see [Chapter 9.3.2 "Conditional Command Execution", page 294](#).

Repeatedly Executed Calls

sub_call_repeat, **sub_call_abs_repeat**, **list_call_repeat** and **list_call_abs_repeat** can be used to automatically execute the body of a subroutine several times.

Index Management and Defragmentation

Index Management

To duplicate, renumber or convert indexed subroutines, **copy_dst_src** is provided.

By **copy_dst_src**, an indexed subroutine is indexed one more time. **copy_dst_src** only alters the corresponding entries in the internal management table and does not modify the list memory content.

No longer needed indices (unneeded entries in the internal management table) can be deleted by **load_sub** directly followed by a **list_return**. Here, too, deletion occurs only in the internal management table, while the list commands of the previously indexed subroutine continue to reside in the list memory.

get_sub_pointer can be used to query whether a subroutine for a particular index exists. If no subroutine exists, **get_sub_pointer** returns a “-1” value (that is, $2^{32}-1$).

A true duplicate of an indexed subroutine in the protected list memory area “List 3” can be created (after **copy_dst_src**) with **save_disk/load_disk**.

Subroutines with multiple indices are thereby written several times to the list memory. Keep this in mind in order to prevent unintended memory overflow of the protected list memory area “List 3”.

load_sub always enters a new indexed subroutine after the indexed subroutine with the highest memory address. Therefore, subroutines that are no longer required and are located in the protected list memory area “List 3” may block memory positions for further indexed subroutines.

For this reason, simply increasing the size of the protected list memory are “List 3” by **config_list** fails to produce further usable memory for storing additional indexed subroutines (the protected list memory area “List 3” can only be expanded downward, not upward).

Defragmentation

This situation can be resolved by defragmenting with **save_disk/load_disk**.

Thereby, all indexed subroutines and (by **set_sub_pointer**) subsequently indexed subroutines are rewritten to the protected list memory area “List 3” (in index sequence, starting at the lowest memory position of the protected list memory area “List 3”).

The now-available upper memory positions can then be used for storing additional indexed subroutines.

Notes

- Before calling **load_disk**, be sure the protected list memory 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**. **save_disk** returns the number of stored list commands. No-longer-needed subroutines should previously deleted from the index management by a **load_sub** which is directly followed by a **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 **list_call**.
- **save_disk** stores subroutines starting from the start address to the first-encountered **list_return**. Relative jumps are not evaluated. So do not use branches to several **list_return**. Instead, reclose eventual branches prior to one single **list_return**.

Subsequent Protection and Conversion of Non-Indexed Subroutines

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

There are basically two methods to protect non-indexed subroutines subsequently:

(1) Changing the configuration

The part of the list memory area in which the non-indexed subroutine has been written is assigned to the protected list memory area "List 3" by **config_list**. The subroutine subsequently protected in this manner remains non-indexed (with unaltered memory address) and can, as before, be called by **list_call**.

(2) Converting to indexed subroutines

set_sub_pointer is used to index a non-indexed subroutine and thus include it in the memory management of the indexed subroutines.

By **save_disk/load_disk**, it can subsequently be copied as an indexed subroutine to the protected list memory area "List 3", see [Section "Defragmentation", page 114](#).

With a subsequent call by **sub_call** via the index, the subroutine in the protected list memory area "List 3" is then started.

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 list memory area "List 3" are not allowed, see [Chapter 6.5.3 "Jumps", page 119](#).

With converting to indexed subroutines (method 2), also all other programming rules for indexed subroutines must be observed, see [Section "Rules for Programming", page 113](#).

Always try to use only one of the two methods. This avoids unintended data loss in the protected list memory area "List 3" by overwriting.

If you begin working with method 1 but later want to also use indexed subroutines: then you should convert all non-indexed subroutines residing in the protected list memory area "List 3" to indexed subroutines using method 2, before you define the first indexed subroutine by **load_sub**.

When doing so, observe the following Notes.

Notes

- If method 1 is used and you remove overwrite-protection for a part of the protected list memory area "List 3", then you risk overwriting indexed subroutines or previously protected subroutines.
- Non-indexed subroutines subsequently protected with method 1 can under some circumstances be overwritten by a later **load_sub** or **load_disk**.
- **set_sub_pointer** links the supplied index with the specified 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 indexed and no longer callable by the index.

- If using method 2, you should use it fully. If the **set_sub_pointer** 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 list memory area "List 3".
- **save_disk** ignores all non-indexed subroutines, even those subsequently protected in the protected list memory area "List 3" by method 1. Be aware that they can be overwritten there by **load_disk**.
- **save_disk/load_disk** automatically replaces unallowed commands (for example,, **set_end_of_list**) with **list_nop** commands.
- Indexed subroutines repeatedly indexed with **copy_dst_src** are duplicated in the list memory at a subsequent **save_disk/load_disk**. This can result in a memory overflow in the protected list memory area "List 3".
- Before executing **load_disk**, be sure the protected list memory 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.

Unprotecting Subroutines

The protection of a subroutine stored in the protected list memory area "List 3" is removed, if it is assigned by **config_list** to one of the list memory area "List 1" or "List 2".

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

6.5.2 Character Sets and Text Strings

For marking tasks, it is convenient to use the RTC6 list memory for storing command lists as separate subroutines that define how the scan system should mark the needed characters and/or text strings.

To simplify management of characters and text strings, the RTC6 PCIe Board provides the possibility of storing indexed character definitions and text string definitions in its protected list memory 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 RTC6 PCIe Board management table – separately from indexed subroutines.

The individual character and text string definitions must specify the shape and orientation (for example, 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 end position of a character or a text string 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 list memory area "List 3" by [load_char](#) (the resultant automatically-assigned memory address can be queried by [get_char_pointer](#)). Alternatively, a non-indexed subroutine can be subsequently indexed with [set_char_pointer](#) and then copied by [save_disk/load_disk](#) as an indexed character in the protected list memory area "List 3".

The RTC6 PCIe Board manages up to 4 character sets, each with 256 indexed characters.

Other than that, the same rules as for indexed subroutines are applicable, see [Section "Indexed Subroutines", page 112](#) and [Section "Subsequent Protection and Conversion of Non-Indexed Subroutines", page 115](#).

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](#) and [mark_serial](#). Otherwise, improper marking might occur during execution of the indexed character.

Calling Indexed Characters

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

To label serial numbers, indexed characters (digits) can also be called up with **mark_serial**, see [Chapter 7.5 "Marking Dates, Times and Serial Numbers", page 209](#).

The marking of entire text passages can be started by **mark_text** (or the "AbsCall" command **mark_text_abs**). The desired character set can be selected in advance by **select_char_set**.

When a **mark_text** 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** in the list memory. Make sure that no unwanted memory overflow of the respective memory area occurs.

Defining Indexed Text Strings for Time, Date and Serial Number

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**.

With **load_text_table**, a sequence of list commands defining a text string can be loaded directly into the protected list memory area "List 3" 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 indexed with **set_text_table_pointer** and then copied by **save_disk/load_disk** as an indexed text string in the protected list memory area "List 3".

The RTC6 PCIe Board manages up to 42 indexed text strings.

Other than that, the same rules as for indexed subroutines are applicable, see [Section "Indexed Subroutines", page 112](#) and [Section "Subsequent Protection and Conversion of Non-Indexed Subroutines", page 115](#).

Notes

- **set_char_table** is synonymous with **set_text_table_pointer**.

Calling Indexed Text Strings

Indexed text strings can be called for marking times, dates and serial numbers by `mark_time`, `mark_date` and `mark_serial` (or the “AbsCall” commands `mark_time_abs`, `mark_date_abs` and `mark_serial_abs`), see [Chapter 7.5 “Marking Dates, Times and Serial Numbers”, page 209](#).

Managing Indexed Characters and Text Strings

The index management of indexed characters and indexed text strings occurs separately from the index management of indexed subroutines.

Index management by users (renumbering, duplicating, ...) resembles index management of indexed subroutines, see [Section “Index Management and Defragmentation”, page 114](#), using `copy_dst_src`, `load_char`, `load_text_table`, `get_char_pointer`, `get_text_table_pointer` and `save_disk/load_disk`. Defragmentation of the protected list memory area “List 3” also includes indexed characters and text strings.

6.5.3 Jumps

`list_jump_pos` (synonymous with `set_list_jump`) and `list_jump_rel` allow the definition of jumps to a specified address which are carried out by the RTC6 PCIe Board at runtime.

With `list_jump_pos`, an absolute memory address within the configured list memory area (“List 1” and “List 2”) can be specified. Jumps into and out of the protected list memory area “List 3” are not allowed with `list_jump_pos`. A `list_jump_pos` having such an unallowed jump address is ignored during execution.

With `list_jump_rel`, jump distances (that is, relative memory addresses) can be specified. `list_jump_rel` can be used in all list memory areas, even the protected list memory 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, `list_jump_rel` is ignored by the RTC6 PCIe Board during processing.

If `list_jump_rel` is used in an indexed subroutine, you must further ensure the jump does not exceed the boundaries of the subroutine. During processing of indexed subroutines, relative jumps that exceed the boundaries of a subroutine are ignored by the RTC6 PCIe Board.

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 RTC6 PCIe Board ignores the corresponding **Jump command** – hence, the user 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 [Chapter 9.3.2 "Conditional Command Execution", page 294](#).
- **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, for example, to wait for confirmation of a signal.

6.5.4 RTC4-Circular Queue Mode

With the RTC6 PCIe Board, the RTC4 circular queue mode does not exist.

Nevertheless, users can actually replace this operational mode with the RTC6 PCIe Board by using an alternating list change and **load_list**.

`load_list (3, 0)` ensures that new commands are loaded only into an already processed list (that is not **BUSY** list execution status), without needing to explicitly specifying the number of the list, see also [Section "Alternating List Changes", page 110](#) and [Section "Loading with Protection", page 105](#).

6.5.5 Loops

Although list jumps, see [Chapter 6.5.3 "Jumps", page 119](#), and conditional jumps, see [Chapter 9.3.2 "Conditional Command Execution", page 294](#), let you repeat any number of list commands endlessly or under external control, precisely specifying the number of executions is not always reliably possible.

But this can be achieved by 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.

list_repeat and **list_until** must always be used in pairs. Unpaired or supernumerous 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 (for example, **list_repeat** directly followed by **list_until**) terminate immediately and are not repeated.

The command pairs can be located both within lists and within subroutines.

Within subroutines, **list_until** performs a **list_jump_rel** to the address directly after the associated **list_repeat**. Loops do not function beyond the boundaries of a subroutine, because list jumps into or out of subroutines are not allowed, see [Chapter 6.5.3 "Jumps", page 119](#).

Within lists, however, **list_until** executes a **list_jump_pos** (to the address directly after the associated **list_repeat**). Thus, **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 has been terminated (as may be the case 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 finds an associated **list_repeat**.

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

Explicit list jumps into or out of the body of a **list_repeat/list_until** loop are allowed because they 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).

If a **list_repeat/list_until** loop is to be executed with an initially unknown number of repetitions, a high value (for example, greater than the highest expected number) can be specified for the **Number** parameter of **list_until**. Within the loop, a conditional branch (for example, which is dependent on an external signal) can jump to a position outside the loop and leave the loop this way. At this point there should be a **list_until(Number = 0)** to end the just left loop properly.

6.6 Using Several RTC6 PCIe Boards in One PC

As many RTC6 PCIe Boards as the PCIe bus allows can be operated simultaneously in one PC. However, the **RTC6 DLL** internal card management can manage a maximum of 255 RTC6 PCIe Boards and RTC6 Ethernet Boards at the same time, see [Chapter 16.5.3 "About the RTC6 Board Management", page 887](#).

All RTC6 PCIe Boards work independently of each other. The command lists of all boards can be loaded and executed at any time.

The RTC6 command set allows two methods to write user program when using several RTC6 PCIe Boards:

- ["Multi-Board Programming", page 122](#)
- ["Single-Board Programming", page 123](#)

6.6.1 Multi-Board Programming

In this programming method, the multi-board versions (command names with prefix “**n_**”) of the RTC6 commands are used.

Compared to single-board commands (command names without prefix “**n_**”), the board number⁽¹⁾ to which the command is to be transmitted must be specified before the parameters. All other parameters are identical.

The installed RTC6 PCIe Boards are numbered in the order found during initialization (starting with 1), see [Chapter 16.5.3 "About the RTC6 Board Management", page 887](#).

The multi-board command **n_get_serial_number** can be used to determine which RTC6 PCIe Boards have been assigned numbers. See also example (3) below.

rtc6_count_cards returns the number of RTC6 PCIe Boards in the RTC6 board management.

Notes

- Multi -board commands are sent to the active board (default board), if the specified number is > 255 or 0 (real boards begin at 1).
- If no real card is entered in the RTC6 board management under the specified number, the Multi -board command is rejected.
- All multi-board commands are listed in [Chapter 10.1 "Overview", page 301](#). for nearly all single-board commands a corresponding multi-board command is available. Exceptions are explicitly noted in the command descriptions (in [Chapter 10.2 "RTC6 Command Set", page 314](#)).

Examples (Pascal)

(1) Write a **Jump command** to the point (500, 500) into the current list of RTC6 PCIe Board #1:

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

(2) Process list with number **list_no** (1 or 2) on the RTC6 PCIe Board with the number specified by the variable **RTC6_no**:

```
n_execute_list(RTC6_no, list_no)
```

(3) Return the serial number of RTC6 board #1:

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

(1) As an unsigned 32-bit value.

6.6.2 Single-Board Programming

During single board programming, one of the inserted RTC6 PCIe Boards is defined with `select_rtc` as default card. All single board commands following `select_rtc` are sent to the defined board until `select_rtc` is called once again.

Multi-Board commands are not influenced by `select_rtc` (if the card number is valid, see above).

Care must be taken if a process uses multiple boards by multiple threads, because `select_rtc` is not thread-specific but board-specific. It immediately redirects the output of *all* currently running threads of a process to the specified RTC6 PCIe Board.

Notice!

- `select_rtc` defines the active RTC6 PCIe Board for all threads of one process (user program) that are currently running. In multi-threaded user programs, this can result in programming errors.

6.6.3 Master/Slave Operation

If several RTC6 PCIe Boards are to be operated clock-synchronized, then they must be connected pairwise with each other by the Master and Slave connectors and installed in preferably (recommended) adjacent PCIe slots. Connect the Master connector of one board to the Slave connector of another board. Suitable connection cables, see [Figure 8](#), are available from SCANLAB.

An RTC6 PCIe Board automatically gets the master board of a master/slave chain, if a further RTC6 PCIe Board is connected to its Master connector but no further RTC6 PCIe Board is connected to its Slave connector. All other RTC6 PCIe Boards are slave boards. See also [Chapter 4.4 "Master Socket Connector, Slave Socket Connector", page 65](#).

`get_master_slave` can be used to query separately for each RTC6 PCIe Board the master/slave status, that is, whether it is operated as a master, slave or single board.

For a source code example on how to check which RTC6 PCIe Board is the master and which one is slave, see [Section "Example Code \(Delphi\)", page 126](#).

After synchronization, the clock phase of each board is delayed by 0...3 1/64 μ s clock cycles (= approx. 0 ns...50 ns) in relation to the clock phase of the preceding (upstream) board.

Without synchronization, delays of up to 10 μ s 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).

Initialization with RTC6 Software Package $\geq 1.5.0$ (\geq RBF 619)

On all RTC6 PCIe Boards of a master/slave chain must have been executed:

- `load_program_file`

Clock Phase Synchronization

The synchronization takes place automatically as soon as 2 RTC6 PCIe Boards are connected.

`sync_slaves` does not have to be called anymore.

The synchronization status can be queried at any time with `get_sync_status` even without a prior call of `simulate_ext_start`.

If an RTC6 PCIe Board has been synchronized to external clock cycles in the meantime (see [Chapter 7.4.10 "Synchronization of the RTC6 Clock Cycle and an External Clock Signal", page 206](#)), it is automatically resynchronized with the master board upon leaving this state.

When `load_program_file` is executed, the synchronization of all subsequent slave boards is lost for a short time. However, it is automatically reestablished.

Notes

- The master board does not pass 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 ns...50 ns clock phase shift.
- The correction files and lists must be loaded separately onto all RTC6 PCIe Boards.
- By `get_sync_status(Bit #21...Bit #30)`, the exact propagation time in 1/64 μ s clock cycles between two RTC6 boards (outbound and return) can be read out.

Initialization with RTC6 Software Package $< V1.5.0$ (\leq RBF 618)

On all RTC6 PCIe Boards of a master/slave chain must have been executed:

- `load_program_file`
- `load_correction_file`

The synchronous timing with stable phase position of a master/slave chain is severed by the first not-initialized board. If an RTC6 PCIe Board is initialized by `load_program_file` but connected as slave to a board which has not been initialized by `load_program_file`, then it is subject to its own clock with a random phase position.

Clock Phase Synchronization

If the RTC6 PCIe Boards of a master/slave chain are to be synchronously clocked with a defined relative clock phase, then the boards must be correspondingly synchronized by `sync_slaves`.

For this, it is necessary to send `sync_slaves` one-time to the master board only. 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`.

Notes

- The master board does not pass 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 ns...50 ns clock phase shift.
- If a board in a synchronized master/slave chain is externally clocked separately by a cycle synchronization, then this clocking refers exclusively to this board. All other boards in the master/slave chain continue to be synchronized with the original clock of the master board. If cycle synchronization is then deactivated again, the affected card remains asynchronous. It can only be synchronized again by calling `sync_slaves` once more.
- The correction file and lists must be loaded separately onto all RTC6 PCIe Boards.



Synchronous Starts and Synchronous Stops

Within a master/slave chain, **External Starts** (if enabled with `set_control_mode`) and **External Stops** are passed on:

- from one board to all boards

Therefore, it can be triggered:

- A synchronous start of all synchronized boards within a master/slave chain (with presetable track delays) can be triggered by an external start signal, a `simulate_ext_start` or a `simulate_ext_start_ctrl` at any board
- A synchronous stop of all synchronized boards within a master/slave chain by an external stop signal or a `simulate_ext_stop` at any board

In contrast, *not* passed on are:

- Starts by `execute_list`
- Starts by `execute_at_pointer`
- Stops by `stop_execution`

Therefore, these must be separately executed even at master/slave-synchronized boards.

Notes

- See also [Chapter 9.3.1 "Starting and Stopping Lists by External Control Signals and Master/Slave Synchronization"](#), page 288.
- With `master_slave_config` the master/slave interface properties can be configured individually for each RTC6 board.



Example Code (Delphi)

The following example Delphi source code shows how to check which RTC6 PCIe Board is the master and which one is slave.

The code must be included in a user program, see [Chapter 6.2.5 "Example Code \(C\)", page 99](#).

```
if init_RTC6_dll() <> 0 then halt;           // Initialize the RTC6 DLL
if RTC6_count_cards() <> 2 then halt;         // Are 2 RTC6 in the PC?

for CardNo := 1 to 2 do                      // Load program and 3D correction files onto both boards
begin
  // Stop RTC6 if a task is currently still running. Optional:
  // n_stop_execution(CardNo);
  if n_load_program_file(CardNo, nil) <> 0 then halt;
  if n_load_correction_file(CardNo, nil, 1, 2) <> 0 then halt;
end;

// Check for Errors (Bit #12...Bit #15)
if (n_get_sync_status(1) and $F000 = 0) and (n_get_sync_status(2) and $F000 = 0) then
begin
  // 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
end else
halt;

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);
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); // for ≤ RBF 618
Result = n_get_sync_status(Master) and $3FF; // must be 640
Result = n_get_sync_status(Slave) and $3FF; // must be <4

// MasterCfg, SlaveCfg must be defined elsewhere, see manual
n_master_slave_config(Master, MasterCfg);
n_master_slave_config(Slave, SlaveCfg);
```

6.7 Usage of RTC6 PCI Express Boards by Several User Programs

Usage of RTC6 PCIe Boards by several user programs is coordinated by the (RTC6 DLL-internal) RTC6 board management, see also [Chapter 16.5.3 "About the RTC6 Board Management", page 887](#). By `init_rtc6_dll`, it is initialized.

`init_rtc6_dll` automatically grants a user program access rights (by `acquire_rtc`) to the found boards, as long as the access right has not been already assigned to another user program (several RTC6 PCIe Boards or user programs can be used simultaneously, but no board can be simultaneously used by several user program).

Access rights (even if temporary) to boards are granted on an exclusive basis by the [RTC6 DLL](#).

Multiple threads of *one* user program can use the same board, but can not send commands to it at the same time (the [RTC6 DLL](#) automatically serializes the command calls).

Without access rights, a board can only be accessed by a user program through purely [RTC6 DLL](#)-internal commands that do not require access rights (for example, `get_error`, `get_last_error` and `select_rtc`).

If a user program has granted access rights for a board, then this board can be acquired by another user program only after the original user program explicitly has been released it explicitly by `release_rtc` or `free_rtc6_dll`.

When a board is acquired by `acquire_rtc` (or `init_rtc6_dll` or `select_rtc`), a version check of the [RTC6 DLL](#), `RTC6OUT.out`, and `RTC6RBF.rbf` is performed.

If these files are not loaded yet, then a version check cannot be explicitly performed but the check is still regarded as successful and does thus not hinder the board acquisition. If these files are loaded and the version check detects an error, then access is denied (`get_last_error` return code `RTC6_ACCESS_DENIED` | `RTC6_VERSION_MISMATCH`).

6.7.1 Notes on Board Acquisition by a User Program

`init_rtc6_dll`, `acquire_rtc`, `free_rtc6_dll`, `release_rtc` and `select_rtc` affect the access rights of RTC6 PCIe Boards. They do not initialize the RTC6 PCIe Boards.

A user program acquiring a board by `acquire_rtc` (or `init_rtc6_dll` or `select_rtc`) inherits its unadjusted memory contents and operational state. The user program therefore can use the stored data and settings of the board and could intervene in the flow of any list 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 RTC6 PCIe Board can be further used without changes, because in this situation all [RTC6 DLL](#) configuration data remain unaltered. However, the above does not apply, if the acquired board has been released by `free_rtc6_dll` and reacquired after `init_rtc6_dll`.

When an RTC6 PCIe Board is acquired by another user program, some important information managed by the previous user program only in the **RTC6 DLL** is *not* (automatically) taken over. 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 initialization by **load_program_file** and all settings are newly defined anyway, such missing information would not be relevant.

On the other hand, if the acquiring user program is supposed to further use the inherited state of the RTC6 PCIe Board, then it must explicitly query the missing **RTC6 DLL** information, receive it from the previous user program and explicitly re-establish it so that **RTC6 DLL** and board remain consistent. In this regard, observe the following notes.

Notes

- For a correct behavior of the input pointer at the list borders, the memory configuration currently set in the **RTC6 DLL** for the acquiring user program must be consistent to the current memory configuration of the acquired board. **get_config_list** obtains the current memory configuration of the board and sets it in the **RTC6 DLL** correspondingly.
- The management tables of protected functions (indexed subroutines, character sets and text strings) are located on the board. 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 **Section "Index Management and Defragmentation"**, page 114.

An alternative restoration method is not possible. The intermixed loading of protected functions by differing applications should therefore be avoided.

- The input pointer is generally not inherited (the input pointer location currently saved in the **RTC6 DLL** for the acquiring user program is used, maybe corrected after **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 (in regards to **LOAD list status** and **READY list status**) 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 **RTC6 DLL**. Though settings used by the previous user program can not generally be queried, new settings can of course be set 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

In order to catch errors in the program flow, the RTC6 PCIe Board carries out a general error handling. With some commands there is also an individual error handling.

General errors occur, for example, if:

- the user program has no access rights for the board (`RTC6_ACCESS_DENIED`)
- the board fails to respond to a control command (`RTC6_TIMEOUT`)
- PCI communication problems occur during sending (`RTC6_SEND_ERROR`)

Individual errors occur, for example, if:

- calling a command with an unallowed (uncorrectable) parameter (`RTC6_PARAM_ERROR`, see, for example, `get_value` or `write_da_x`)
- rejected sending of a list command (`RTC6_REJECTED`, for example, due to an invalid input pointer)
- sending a control command at an improper time (`RTC6_BUSY`, for example, `goto_xy`, when a list is still being processed)

In such cases, the control commands are not executed and list commands are typically each replaced by `list_nop` (for example, for `RTC6_PARAM_ERROR` or `RTC6_IGNORED`, see `set_end_of_list` as an example).

The bits assigned to these errors are set or accumulated in the **RTC6 DLL** with each command in the board-specific error variables:

- `LastError`
 - Error code
 - Is automatically reset at the beginning of every command
 - Therefore, is a listing of occurred errors from the most recently executed command
 - Can be queried by `get_last_error`⁽¹⁾
- `AccError`
 - Cumulative error code
 - Is reset when initializing the **RTC6 DLL**
 - Can be reset by the user program itself by `reset_error`
 - Are all accumulated error bits since the last error bit reset by `reset_error`
 - Can be queried by `get_error`⁽¹⁾

Error handling takes place separately:

- for each board
- 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.

Error handling only takes place during processing of commands within the **RTC6 DLL** and when sending to the RTC6 PCIe Board. Error handling takes place during execution of a list program.

An example code of how to incorporate board-specific error variables is provided in the command description of `get_error`.

(1) The described mechanism only applies for commands that establish communication with the RTC6 PCIe Board. Commands that *do not* establish communication with the RTC6 PCIe Board (for example, `rtc6_count_cards`, `set rtc4_mode` or `get_serial_number`) neither generate nor alter `LastError` or `AccError` (see also comments in the corresponding command descriptions).



Some control commands (for example, `init_RTC6_dll`, `load_correction_file` or `load_program_file`) additionally return a special error code as the result value that is not buffered and must therefore be immediately evaluated or discarded by the user program.

6.8.1 Download Verification

Verification of RTC6 communication is vital particularly in medical applications. For this purpose, you can activate download verification separately for each board by `set_verify`.

However, this automatically results in extended download times.

If download verification is activated and an error is found, then the error code `RTC6_VERIFY_ERROR` is set, which can be queried by `get_last_error` or `get_error`. Certain operations might immediately be aborted and the board would then no longer be functional (for example, if `load_program_file` has been aborted).

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

(1) 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 RTC6 PCIe Board itself is checked; automatic parameter adjustments (for example, clipping) are not taken into account.

(2) 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.

(3) `load_program_file`

For sending of `load_program_file`, the following is checked:

- `RTC6DAT.dat` is tested by a checksum for file correctness and PCI-transfer correctness.
- `RTC6RBF.rbf` is only checked by a bitwise transfer handshake. No other checking is possible.
- Each loaded section of `RTC6OUT.out` is immediately read back for checking. If an error is detected, then the loading process aborts.



(4) 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 command description of `set_verify` and `verify_checksum`.

(5) Loading of tables

For loading other tables (for example, by `load_varpolydelay`), the transfer is checked for correctness by an immediate read back of the table. In addition to the `get_last_error` return code `RTC6_VERIFY_ERROR`, the corresponding error return value of the loading command is also get set.

6.8.2 Checking for OVERRUNS

By `get_overrun`, it can be checked whether overruns of the 10 μ s clock cycle have occurred. See also Section "Clock OVERRUNS", page 181.



6.8.3 Example Code (C)

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

- An error occurs during initialization with **init_rtc6_dll** (for example, if no RTC6 PCIe Board has been detected)
- The desired RTC6 PCIe Board (here: the board with serial number 12345) is not found
- Access is denied to the desired RTC6 PCIe Board
- An error occurs during **load_program_file** (for example, a version mismatch, file or system error)

The code must be included in a user program, see [Chapter 6.2.5 "Example Code \(C\)", page 99](#).

```
UINT ErrorCode;
ErrorCode = init_rtc6_dll();
if ( ErrorCode )
{
    // Reading the number of RTC6 boards detected during initialization with init_rtc6_dll
    const UINT RTC6CountCards = rtc6_count_cards();
    if ( RTC6CountCards )
    {
        // Detailed error analysis for all detected boards
        UINT AccError( 0 );
        for ( UINT i = 1; i <= RTC6CountCards; i++ )
        {
            // Errors which occurred during execution of init_rtc6_dll
            const UINT Error = n_get_last_error( i );
            if ( Error != 0 )
            {
                AccError |= Error;
                const UINT SerialNumber = n_get_serial_number( i );
                printf( "RTC6 board number %d (serial number %d): Error %d detected\n",
                        i, SerialNumber, Error );
                n_reset_error( i, Error );
            }
        }
        if ( AccError )
        {
            free_rtc6_dll();
            return;
        }
    }
}
```



```
else
{
    printf( "Initializing the DLL: Error %d detected\n", ErrorCode );
    free_rtc6_dll();
    return;
}
else
{
    // Reading the internal board number for the desired RTC6 board
    const UINT SerialNumberOfDesiredBoard ( 12345 );
    const UINT RTC6CountCards = rtc6_count_cards();
    UINT InternalNumberOfDesiredBoard ( 0 );
    for ( UINT i = 1; i <= RTC6CountCards; i++ )
    {
        if ( n_get_serial_number( i ) == SerialNumberOfDesiredBoard )
        {
            InternalNumberOfDesiredBoard = i;
        }
    }
    if ( InternalNumberOfDesiredBoard == 0 )
    {
        printf( "RTC6 board with serial number %d not detected.\n", SerialNumberOfDesiredBoard );
        free_rtc6_dll();
        return;
    }
    // Selecting the desired RTC6 board as the active RTC6 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 ) // RTC6_VERSION_MISMATCH
        {
            if ( ErrorCode = n_load_program_file( InternalNumberOfDesiredBoard, 0 ) )
            {
                printf( "n_load_program_file returned error code %d\n", ErrorCode );
            }
        }
        else
        {
            printf( "No access to RTC6 board with serial number %d\n", SerialNumberOfDesiredBoard );
            free_rtc6_dll();
            return;
        }
        if ( ErrorCode )
        {
            printf( "No access to RTC6 board with serial number %d\n", SerialNumberOfDesiredBoard );
            free_rtc6_dll();
            return;
        }
        else
        {
            // if n_load_program_file has been successful, select the desired board
            (void) select_rtc( InternalNumberOfDesiredBoard );
        }
    }
}
```



6.9 Miscellaneous

6.9.1 Free Variables

8 so-called “free” variables are available.

Users can freely assign data to them by the control command `set_free_variable` and the short list command `set_free_variable_list`.

These variable values can be:

- outputted at the `McBSP interface`, see also [Chapter 9.1.7 “McBSP Interface”, page 286](#)
- read back by `get_free_variable` and `get_value` (see command descriptions)
- recorded by `set_trigger`/`set_trigger4` (see command descriptions)

The free variables let you, for example, transmit control commands over the `McBSP interface` to user hardware or document the operational states of the board (for example, with branches).

Notes

- You can use `set_free_variable` and `set_free_variable_list` to define any unsigned 32-bit values as variable values. However, the `McBSP 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).

7 Basic Functions for Scan Head Control and Laser Control

7.1 Marking Dots, Lines and Arcs

7.1.1 Marking with Vector Commands and "Arc" Commands

As explained in [Chapter 6.1 "RTC6 Software Concept Basics", page 92](#), positioning of the scan system axes (and thus of the laser beam) under RTC6 PCIe Board control is achieved by calling:

- [Jump Commands](#)
- [Mark Commands](#)
- [Arc Commands](#)
- [Ellipse Commands](#)

Each of these commands describes one vector or arc.⁽¹⁾ By using [micro_vector\[*\] Commands](#), arbitrarily shaped trajectories⁽²⁾ can be implemented.

Even numeric and alphabetic characters ultimately consist of the constituent lines, dots and arcs that define them, see [Chapter 7.5 "Marking Dates, Times and Serial Numbers", page 209](#).

[Vector commands \(Jump Commands, Mark Commands\)](#) require as parameters the coordinates of the *end* point of the corresponding vector⁽³⁾. Each vector starts at the *current output position*, which is the end point of the preceding vector or arc.

["Arc" commands \(Arc Commands, Ellipse Commands\)](#) require parameters for the coordinates of the arc center and the arc angle(s). Circular arcs start at the current output position. The elliptical arc start at the position specified by the command parameters. A direct connection to the last output position must be explicitly ensured by the user program itself with suitable parameters.

Otherwise, there is a "Hard jump" there.

The output position after a RTC6 PCIe Board [Hardware reset](#) is the center of the [Image field](#), that is, the point (0|0). Refer to [Chapter 7.3 "Scan Head Control", page 166](#) for a description of the [Image field](#) coordinate system.

At run-time, each vector or arc to be traced by the scan system gets divided by the RTC6 PCIe Board into [Microsteps](#), see [Chapter 7.1.2 "Microstepping", page 139](#)⁽⁴⁾.

[Jump commands](#) serve to move the scan system axes to a new position while the laser is switched *off*.

In contrast, [Mark commands](#) initiate a marking movement while the laser is switched *on* (see also the following description).⁽⁵⁾

To mark a point, [Signals for "Laser Active" Operation](#) must be switched on for the desired time period after a [Jump command](#) or [\[*\]mark\[*\] Command](#), see [Chapter 7.1.3 "Marking Single Dots", page 140](#).

For line and arc marking, the RTC6 PCIe Board automatically switches [Signals for "Laser Active" Operation](#) on at the beginning of a [Mark command](#) and later switches it back off (for example, at the beginning of a subsequent [Jump command](#)).

The synchronization of scan head control and laser control can be adjusted by the user to the respective application by setting delays, see [Chapter 7.2 "Delay Settings – Coordinating Scan Head Control and Laser Control", page 144](#).

Adjustment of laser parameters is described in [Chapter 7.4 "Laser Control", page 183](#).

A thoroughly-commented example code for a basic marking task is shown in [Chapter 7.1.4 "Example Code \(C\)", page 141](#).

(1) Here, wider line widths can be specified by [set_wobble_mode](#).

(2) See [Glossary entry on page 29](#).

(3) 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]".

(4) Only [iDRIVE](#) scan systems (see [Glossary entry on page 26](#)) which are equipped with an appropriate tuning can execute jumps also in [Jump Mode](#), see [Chapter 8.1.5 "Jump Mode", page 216](#).

(5) 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 the mirrors to move from the start point to the end point of a vector.

The **Signals for "Laser Active" Operation** are automatically switched off at the beginning of the vector and remain switched off during the jump, see also [Chapter 7.2.1 "Laser Delays", page 144](#) and [Chapter 7.2.2 "Scanner Delays", page 146](#). The jump speed is defined by `set_jump_speed` and `set_jump_speed_ctrl`.

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

Upon execution of a `[*]mark[*]` Command (`mark_abs` or `mark_rel`⁽¹⁾), the laser focus is moved linearly from the start point to the end point of the vector. The RTC6 PCIe Board automatically turns on the **Signals for "Laser Active" Operation** at the beginning of a `[*]mark[*]` Command, see also [Section "Polylines", page 136](#).

The mark speed is defined by `set_mark_speed` and `set_mark_speed_ctrl`. It can be changed anywhere in a list by `set_mark_speed` or by `set_mark_speed_ctrl`, if no list is currently being processed.

Arc Commands

The **Arc Commands** `arc_abs` and `arc_rel` can be used for marking circular arcs⁽²⁾. The parameters to be specified are coordinates of the arc center and the arc angle. The circular arc starts at the current output position, with angles counted positively and clockwise (contrary to the mathematical definition).

When an arc command is executed, the laser focus is guided along the specified arc at the specified speed. The **Signals for "Laser Active" Operation** are automatically switched on at the beginning of the execution of an arc command, see also [Section "Polylines", page 136](#).

Polylines

If another `[*]mark[*]` Command (or "Arc" command) follows immediately afterward ("Polyline"), the **Signals for "Laser Active" Operation** remain on.

Therefore, a continuous marking is possible by a direct line-up of `[*]mark[*]` Commands (and "Arc" commands).

The **Signals for "Laser Active" Operation** are switched off at the beginning of the (normal) command that follows the last `[*]mark[*]` Command of a Polyline.

See also [Section "EdgeLevel", page 151](#).

(1) For using abs and rel commands, see [Section "AbsCalls", page 113](#). Additionally are available: timed vector commands, see [Chapter 8.9 "Timed Commands", page 273](#), para vector commands, see [Section "Vector-Defined Laser Control", page 205](#) and 3D vector commands, see [Chapter "3D Commands", page 237](#).

(2) For using abs and rel commands, see [Section "AbsCalls", page 113](#). Additionally, timed arc commands are available, see [Chapter 8.9 "Timed Commands", page 273](#).

Ellipse Commands

The RTC6 command set also provides commands for marking elliptical arcs.

Here (unlike marking of vectors or circular arcs), you generally need to call two commands: `set_ellipse` as well as `mark_ellipse_abs` or `mark_ellipse_rel`⁽¹⁾.

By `set_ellipse` the arc shape is specified, see Figure 33:

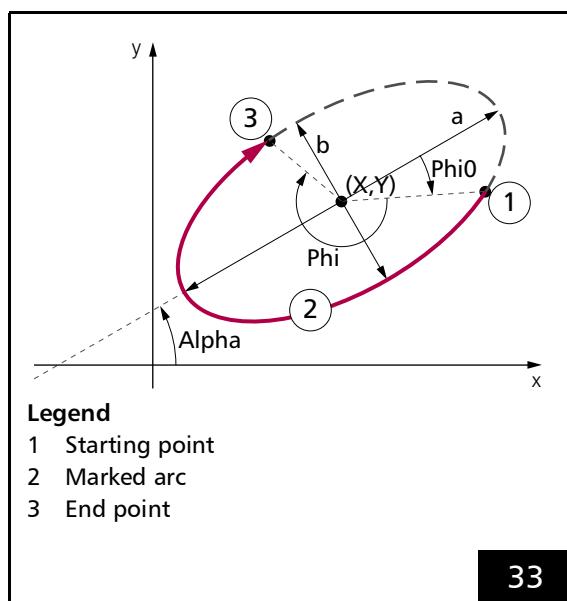
- Lengths a and b of the ellipse half-axes
- The beginning phase angle Phi0 (and thereby the arc starting point position relative to the end point of half-axis a)
- The arc angle Phi (and thereby the length of the to-be-marked ellipse section)

By `mark_ellipse_abs` or `mark_ellipse_rel`, position and orientation of the to-be-executed arc is specified, see Figure 33:

- The coordinates (X, Y) of the ellipse midpoint
- The angle Alpha between the ellipse 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). Phi0 , Phi and Alpha are always relative to axis a .
- Phi0 and Phi are counted positively clockwise (in contrast to mathematical convention). In contrast, Alpha is counterclockwise (in accordance with mathematical convention).
- As with **Mark Commands** and **Arc Commands**, the laser focus moves with the specified mark speed along the specified arc when the **Ellipse command** is executed. The **Signals for "Laser Active" Operation** are automatically switched on at the beginning of an **"Arc" command**, see also **Section "Polylines"**, page 136.
- `set_ellipse` is a short list command, see also **Section "Normal, Short, Variable and Multiple List Commands"**, page 301. Therefore, it can be called between a **Mark command** and an **Ellipse command** without thereby interrupting the **Polyline** (the laser remains on).
- **Ellipse Commands** always begin marking at the starting point determined by the above-mentioned parameters (in contrast to **Mark Commands** and **Arc Commands** which automatically begin marking at the current output position). If the starting point and current position do not match, then a **Hard jump** to the starting point is executed at the beginning of marking (without a **Jump Delay** becoming effective).



33

Marking ellipse-shaped arcs.

(1) For using abs and rel commands, see **Section "AbsCalls"**, page 113.

- Elliptical arcs can also be marked by circular **Arc Commands** (for example, **arc_abs**) if an appropriate coordinate transformation (for example, scaling that differs in the x direction/y direction) has been specified by **set_matrix**. Here, though, the effective mark speed varies along the arc, see also the note on [page 226](#). This contrasts with **mark_ellipse_abs** and **mark_ellipse_rel**, where in $10 \mu\text{s}$ 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 mark speed. For very large eccentricities and also at high mark speeds, however, such stepwise ellipse approximation by a $10 \mu\text{s}$ clock can produce numerical inaccuracies in the end point regions of the large half-axis. Consequently, the effective mark speed there might not be precisely constant (for example, an eccentricity of $a/b = 2$ and 100 **Microsteps** per circumference would produce a speed deviation of approx. 3.7%). However, the outputted point always lies exactly on the ellipse. Moreover, as closed equations do not exist for calculating an ellipse arc length, the step length of the finally-marked **Microstep** is generally shorter and the mark speed correspondingly lower than specified. Nevertheless, the end position is always exact. Likewise, **Sky Writing** might produce run-in/run-out irregularities at the large half-axis. Users themselves must ensure that the parameter values used are consistent with the required precision.

[*]Para[*] Commands

- **para_jump_abs**
- **para_jump_abs_3d**
- **para_jump_rel**
- **para_jump_rel_3d**
- **para_laser_on_pulses_list** (special case)
- **para_mark_abs**
- **para_mark_abs_3d**
- **para_mark_rel**
- **para_mark_rel_3d**
- **timed_para_jump_abs**
- **timed_para_jump_abs_3d**
- **timed_para_jump_rel**
- **timed_para_jump_rel_3d**
- **timed_para_mark_abs**
- **timed_para_mark_abs_3d**
- **timed_para_mark_rel**
- **timed_para_mark_rel_3d**

If the “vector-controlled laser control” is activated, these commands simultaneously vary a signal parameter linearly along the mark or jump vector, see [Section “Vector-Defined Laser Control”, page 205](#).

[*]**para_mark**[*] commands generally do not take **Sky Writing** into account.

7.1.2 Microstepping

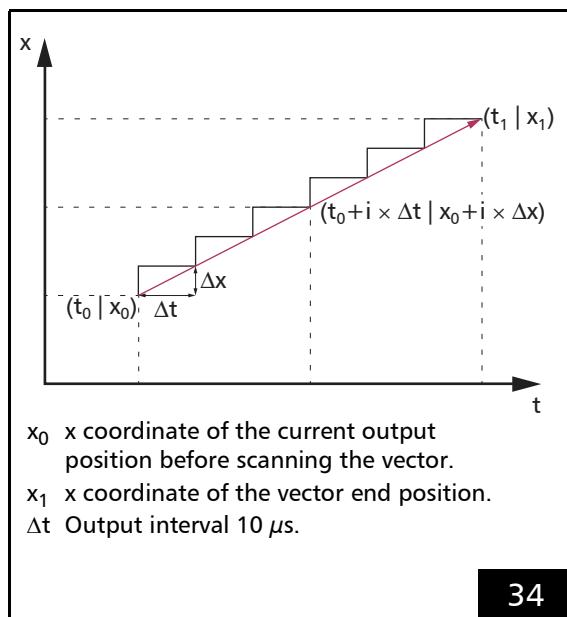
The RTC6 PCIe Board splits up each

- **Jump command**,
- **[*]mark[*] Command** and
- **"Arc" command**

into so-called

- **Microsteps**
(not: **Microvectors**, see **Chapter 8.8 "micro_vector[*] Commands"**, page 272).

The split-up of the x component of a vector into **Microsteps** is shown in **Figure 34**.



The x component of a vector is split-up into **Microsteps**. The y component is split-up in the same way.

All **Microsteps** are transferred to the scan head with a constant output interval (Δt) of 10 μ s and *cannot* be changed.

The following applies:

$$\text{Length } \Delta s \text{ of a Microstep} = v \times \Delta t \text{ }^{(1)}$$

(v = current jump speed or mark speed)

Notes

- Custom curves can be implemented by using **micro_vector[*] commands**, see **Chapter 8.8 "micro_vector[*] Commands"**, page 272.
- Only **iDRI^{VE}** scan systems⁽²⁾ which are equipped with an appropriate tuning can execute jumps also in **Jump Mode**, see **Chapter 8.1.5 "Jump Mode"**, page 216.

(1) Alternatively see **Chapter 8.9 "Timed Commands"**, page 273.

(2) See Glossary entry on **page 26**.



7.1.3 Marking Single Dots

To mark a single point, the [Signals for "Laser Active"](#) [Operation](#) must be switched on for the desired time period, see [laser_on_list](#), [laser_on_pulses_list](#), [para_laser_on_pulses_list](#) and [Chapter 7.4 "Laser Control"](#), page 183.

Alternatively, a single dot can also be marked by a timed [Mark command](#) of length zero, see [Chapter 8.9 "Timed Commands"](#), page 273.



7.1.4 Example Code (C)

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 [Chapter 6.2.5 "Example Code \(C\)", page 99](#).

```
// 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_rtc6_dll();
    return;
}
select_cor_table( 1, 0 ); // assigning correction table #1 to scan head connector #1 (default)

// Laser control initialization, see Chapter 7.4 "Laser Control", page 183.
// Setting the CO2 Mode
set_laser_mode( 0 );

// Setting and enabling the Signals for "Laser Active" Operation
set_laser_control( 0x18 ); // All laser control signals active-LOW (Bit #3 and #4)
    // set_laser_control must be called at least once to activate laser control signals.
    // Later on enable_laser/disable_laser would be sufficient.

// Opens List 1
set_start_list( 1 );

// Setting the standby pulses
set_standby_list( 800, 8 );
    // In RTC4 Compatibility Mode the standby parameters are specified in units of 1/8  $\mu$ s.
    // The RTC6 board multiplies the specified values by 8 to convert them to
    // integer-multiple of 1/64  $\mu$ s. Half of the standby output period = 100  $\mu$ s.
    // Pulse length of the standby pulses = 1  $\mu$ s.

// Timing, delay and speed preset
// Setting the Scanner Delays (see Chapter 7.2.2 "Scanner Delays", page 146).
set_scanner_delays( 25, 10, 5 );
    // Jump Delay = 250  $\mu$ s (specified in [10  $\mu$ s])
    // Mark Delay = 100  $\mu$ s (specified in [10  $\mu$ s])
    // Polygon Delay = 50  $\mu$ s (specified in [10  $\mu$ s])
```



```
// Setting the jump speed and mark speed:  
set_jump_speed( 1000.0 );  
set_mark_speed( 250.0 );  
    // In RTC4 Compatibility Mode the RTC6 board multiplies the speed values by 16.  
    // Jump speed = 1000.0 bits/ms.  
    // Marking speed = 250.0 bits/ms.  
  
// Setting the laser timing, see Chapter 7.4 "Laser Control", page 183.  
set_laser_pulses( 800, 400 );  
    // In RTC4 Compatibility Mode the timing parameters are specified in units of 1/8  $\mu$ s.  
    // The RTC6 board multiplies the specified values by 8 to convert them to  
    // integer-multiple of 1/64  $\mu$ s. Laser HalfPeriod = 100  $\mu$ s.  
    // Laser pulse length = 50  $\mu$ s.  
  
// Setting the Laser Delays, see Chapter 7.2.1 "Laser Delays", page 144.  
set_laser_delays( 100, 100 );  
    // In RTC4 Compatibility Mode the Laser Delays are specified in units of 1  $\mu$ s.  
    // The RTC6 board multiplies the specified values by 32 to convert them to  
    // integer-multiple of 1/64  $\mu$ s.  
    // LaserOn Delay = 100  $\mu$ s.  
    // LaserOff Delay = 100  $\mu$ s.  
  
// Defining the end of the list and the end of command transfer to the RTC6 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 memory 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 RTC6 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.  
// Turning on the laser control signals for 50  $\mu$ s (+ LaserOff Delay - LaserOn Delay):  
laser_on_list( 5 );
```



```
// 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 4 [*]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 a LaserOff Delay, because a Jump command follows.  
  
// 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, see below),  
// a Mark Delay is automatically inserted after the arc command and the laser control signals  
// are automatically switched off after a LaserOff Delay, because a set_end_of_list follows.  
  
// Defining the end of the list and the end of command transfer to the RTC6 board  
set_end_of_list();  
  
// Starting the transferred list  
execute_list( 1 );
```

7.2 Delay Settings – Coordinating Scan Head Control and Laser Control

Scan head control and laser control should suit the dynamic behavior of the system components, that is, the

- response behavior of the laser
- response behavior of the galvanometer scanners ([Tracking error](#))
- the type of interaction between laser radiation and material

The following delays are available for this purpose:

- [Laser Delays](#)
 - [LaserOn Delay](#)
 - [LaserOff Delay](#)
- [Scanner delays](#)
 - [Jump Delay](#) (optionally: variable)
 - [Mark Delay](#)
 - [Polygon Delay](#) (optionally: variable)

7.2.1 Laser Delays

There are two different [Laser Delays](#):

- [LaserOn Delay](#)
- [LaserOff Delay](#)

[Laser Delays](#) determine when the [Signals for "Laser Active" Operation](#) are switched on and off.

As a rule, [Laser Delays](#) have *no* influence on the total marking time. Exceptions:

- A negative [LaserOnDelay](#) value, see [Section "LaserOn Delay", page 145](#)
- Artificially inserted delays, see [Section "Automatic Delay Adjustments", page 154](#)

The [LaserOn Delay](#) and the [LaserOff Delay](#) are set by the undelayed short list command `set_laser_delays`. Their unit is $1/64 \mu\text{s}$ each.

In order to avoid burn-in effects at start and end points of a marking, the laser focus should be moved at a speed which is as constant as possible. Therefore, the laser delay durations must be adjusted to the [Tracking error](#) of the scan head and the set mark speed⁽¹⁾, see also [Chapter 7.2.3 "Notes on Optimizing the Delays", page 154](#).

(1) In addition, automatic readjustment of the laser power during marking can be applied for optimization, see [Chapter 7.4.9 ""Automatic Laser Control""](#), page 196.

LaserOn Delay

The **LaserOn Delay** is automatically inserted at the start of a single **Mark command** and at the start of a series of **Mark command** ("Polyline") and delays the switching on of the laser.

It can be used for several purposes:

- At the beginning of a marking, the mirrors must be accelerated to the specified mark speed (if necessary, from a halt), see [Figure 37](#). This phase can be suppressed with a sufficiently high positive **LaserOn Delay** value until the mirrors have already reached a certain angular speed before the laser is switched on. On the other hand, the **LaserOn Delay** must not be too long, otherwise the first part of the marking is cut off.
- Some materials/applications (for example, welding) take some time until they react as desired to the exposure to laser radiation. In this case, it can be useful to "preheat" the starting point of the marking. This can be achieved by setting a *negative* **LaserOn Delay** value. However, this extends the total marking time because the **LaserOn Delay** is inserted prior to the current **Mark command** as scanner delay.⁽¹⁾

LaserOff Delay

The end of a marking is not defined by the marking itself, but by the first "normal" list command that is not a **Mark command**, for example, a **Jump command**.

The acceleration phase at the beginning of a movement leads to a time difference between the respective set position and the actual position of the mirrors, see [Figure 37](#).

The laser is to be switched off until the *actual value* of the end position is reached (but not already at the *set position*). Therefore, a **LaserOff Delay** is inserted⁽²⁾ automatically after the end of each marking by the first "normal" list command (no "short" list command), see also [Section "Notes", page 148](#). This can be used to compensate for the **Tracking error** of the scan head.

(1) This scanner delay is automatically extended, if a preceding **LaserOff Delay** has not yet been expired, see [Section "Automatic Delay Adjustments", page 154](#).

(2) In **DSP mode < 3** (see **set_dsp_mode**), the following applies for short marking vectors: if a preceding **LaserOn Delay** has not yet been expired, the **LaserOff Delay** is temporarily automatically extended accordingly, see [Section "Automatic Delay Adjustments", page 154](#).

7.2.2 Scanner Delays

There are three different types of **Scanner Delays**:

- **Jump Delays** (optionally: variable)
- **Mark Delays**
- **Polygon Delays** (optionally: variable)

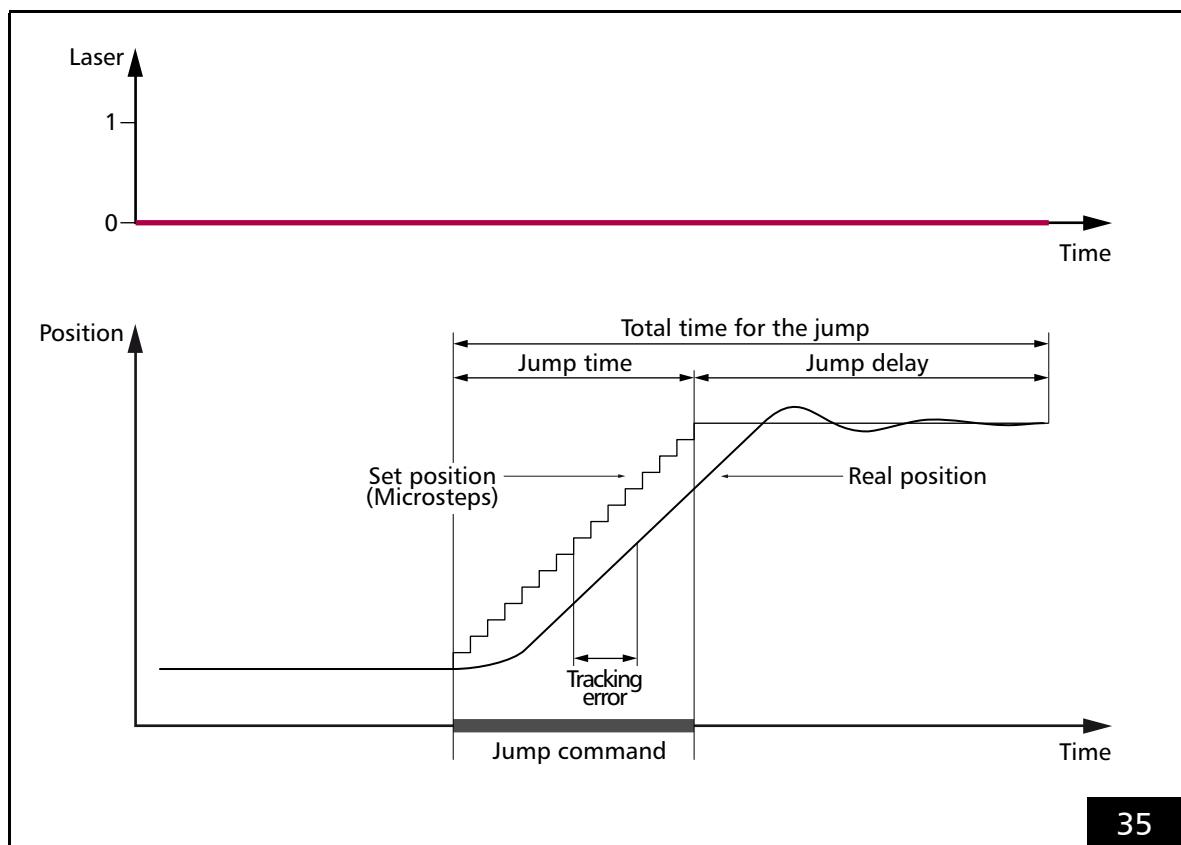
After each **Jump command** and **Mark command**, the RTC6 PCIe Board inserts one of these **Scanner Delays** before the next command is executed (unless otherwise specified).

These **Scanner Delays** are defined by `set_scanner_delays`. Their unit is $10 \mu\text{s}$ each.

Jump Delay

The **Jump Delay** is specified by `set_scanner_delays` with the **Jump** parameter.

A typical course for a single **Jump command** and the belonging **Jump Delay** is shown in Figure 35.



Scan head control during a **Jump command** with a constant **Jump Delay**.
The laser remains off.

Variable Jump Delays

With short jumps, the scan head often does not reach the full jump speed. Then *shorter* Jump Delays are sufficient for settling of the mirrors:

- “Variable Jump Delays”

For this, there is:

- “Variable Jump Delays” mode

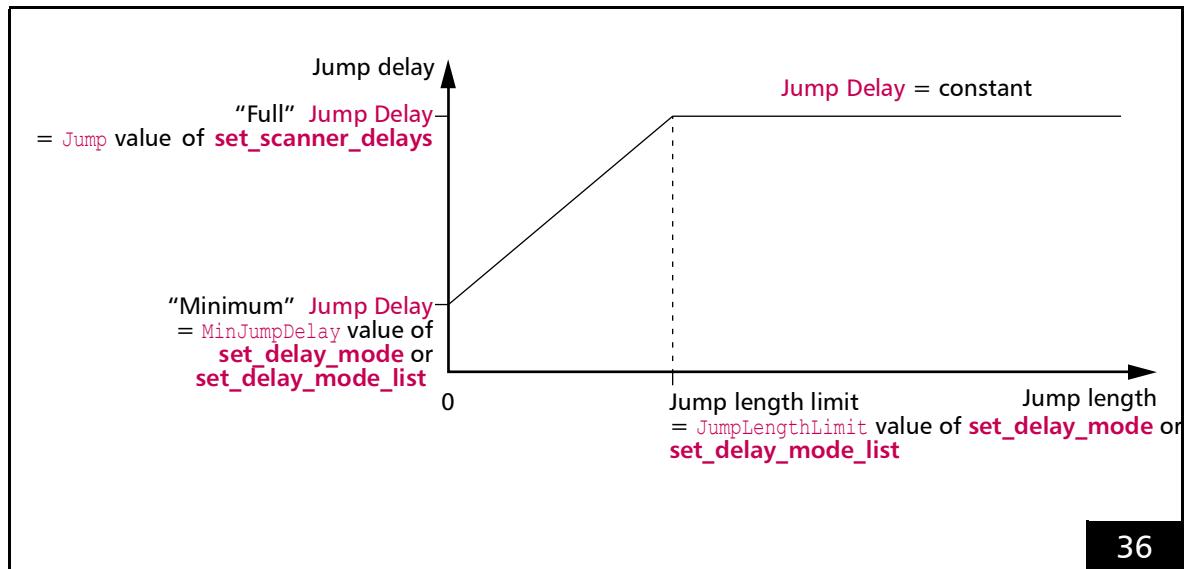
“Variable Jump Delays” mode is switched on by `JumpLengthLimit > 0` (at `set_delay_mode` and `set_delay_mode_list`).

Then the RTC6 PCIe Board inserts a linearly interpolated Jump Delay between `MinJumpDelay` (from `set_delay_mode` or `set_delay_mode_list`) and `Jump` (Jump Delay from `set_scanner_delays`) for jump lengths between 0 and `Jump length limit`, see Figure 36.

With jump lengths larger than `JumpLengthLimit` a “full” Jump Delay is always inserted.

Notes

- With “Variable Jump Delays” mode, total marking time is reduced, especially when there are many short jumps.
- “Variable Jump Delays” mode is switched off by `JumpLengthLimit = 0` (at `set_delay_mode` and `set_delay_mode_list`).
- After jump vectors of length 0, the duration of “Variable Jump Delays” is 0.
- The “minimum” Jump Delay should not be larger than the “normal” Jump Delay. Otherwise, “Variable Jump Delays” can become very large.



“Variable Jump Delays” mode: Jump Delay value depending on the jump length.

Mark Delay

The **Mark Delay** is specified by `set_scanner_delays` with the **Mark** parameter.

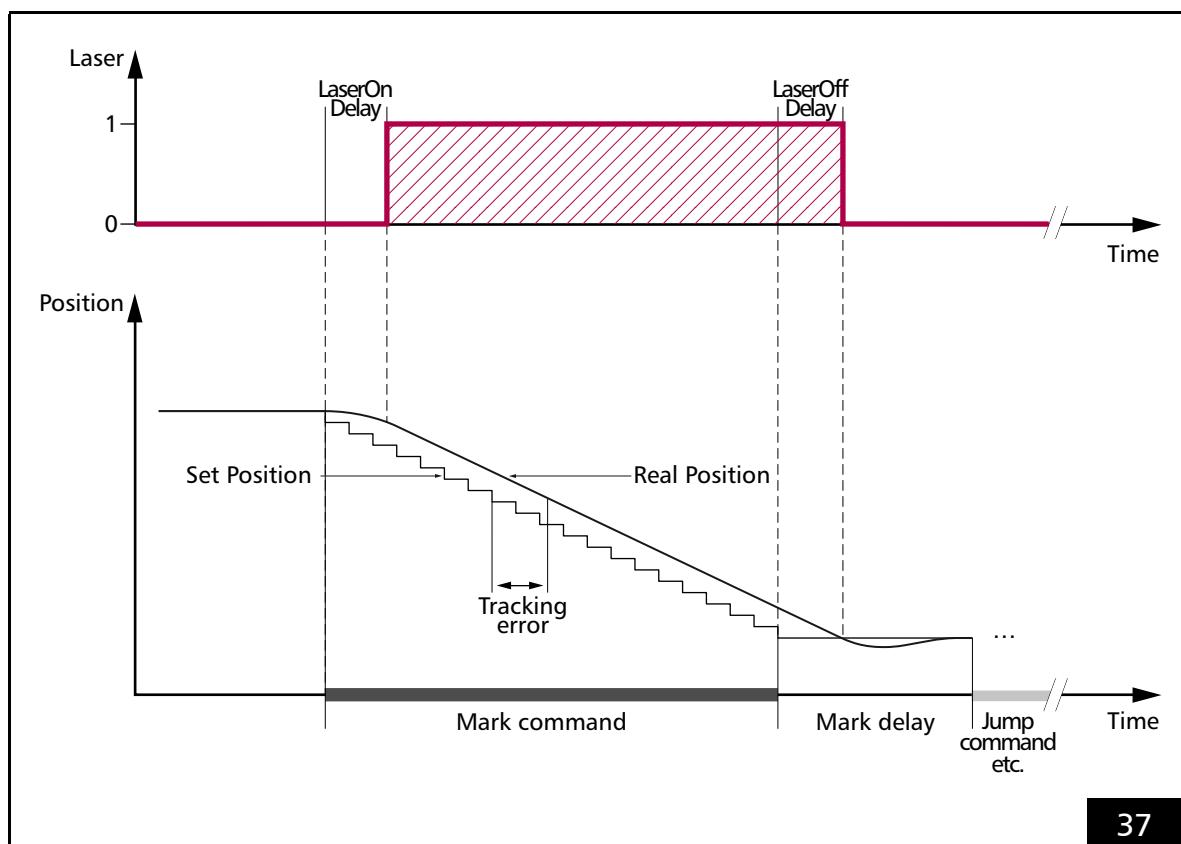
A typical course for a single **Mark command** with the and the belonging **Laser Delays** is shown in Figure 37.

The duration of the **Mark Delay** should suit the dynamic properties of the scan head (**Tracking error**, tuning) and the mark speed set.⁽¹⁾

(1) About the interaction of **Laser Delays** and **Scanner Delays** in DSP mode < 3 (see `set_dsp_mode`), see Section "Automatic Delay Adjustments", page 154.

Notes

- If no further **Mark command** follows a **Mark command**, a **Mark Delay** is inserted automatically and the laser is switched off after a **LaserOff Delay**.
- If a further **Mark command** follows a **Mark command** (= "Polyline"), a (variable) **Polygon Delay** is inserted and the laser remains switched on, see Section "Variable Polygon Delays", page 150.
- Short list commands do not lead to insertion of a **Mark Delay** or **Polygon Delay** and not to a laser switch off.
- Note that a **Mark command** of length 0 corresponds to a **list_continue**, if it is not timed, see Chapter 8.9 "Timed Commands", page 273.



Scan head control and laser control timing during a **Mark command** or "Arc" command with a **Mark Delay**. The laser is on in the hatched period.

Polygon Delay

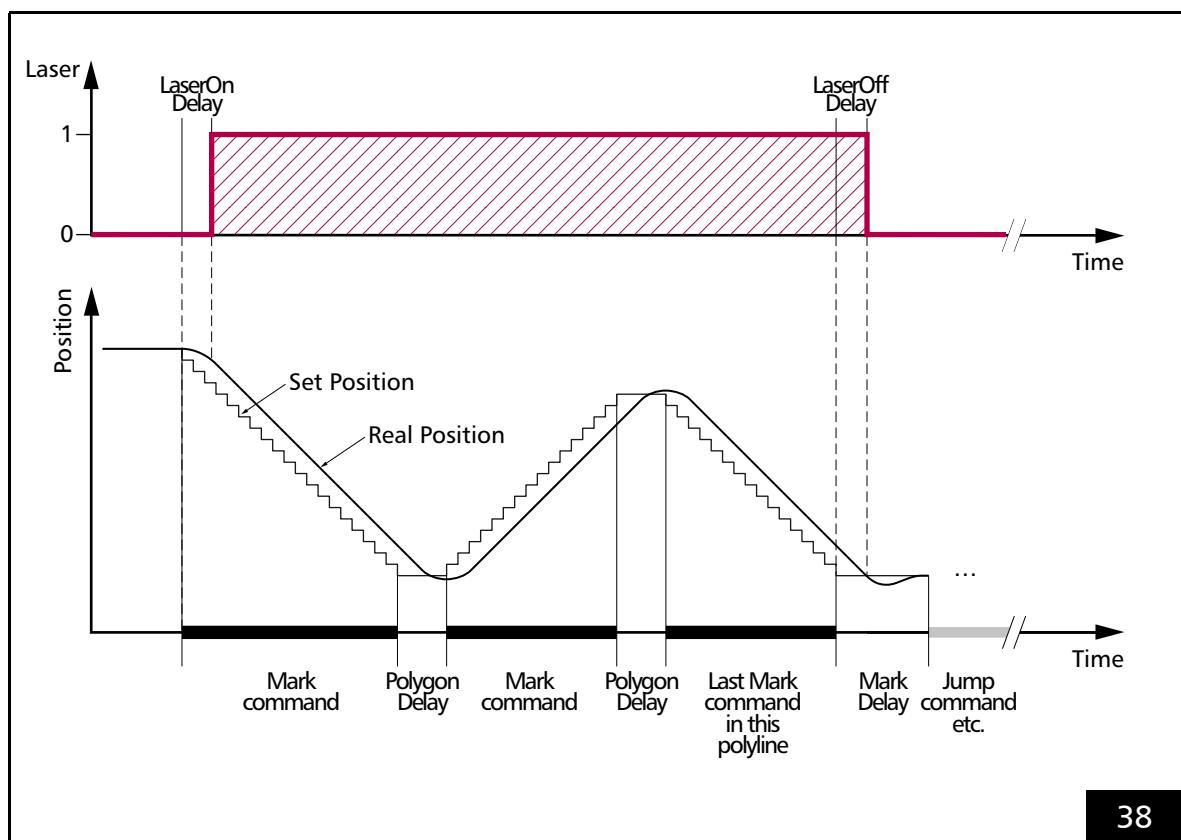
The **Polygon Delay** is specified by `set_scanner_delays` with the **Polygon** parameter.

A typical course for a series of **Mark commands** ("Polyline") where **Polygon Delays** (instead of **Mark Delays**) are inserted between the individual **Mark command** is shown in [Figure 38](#).

Short list commands do not interrupt a **Polyline**.

If the (usually smaller) angles between the markings vary only slightly usually a **Polygon Delay** value can be specified that is smaller than the **Mark Delay** value.

If, on the other hand, the angles vary significantly, then a "Variable **Polygon Delay**" is recommended, see [Section "Variable Polygon Delays", page 150](#).



Scan head control and laser control timing during a **Polyline** with a constant **Polygon Delay**.

Variable Polygon Delays

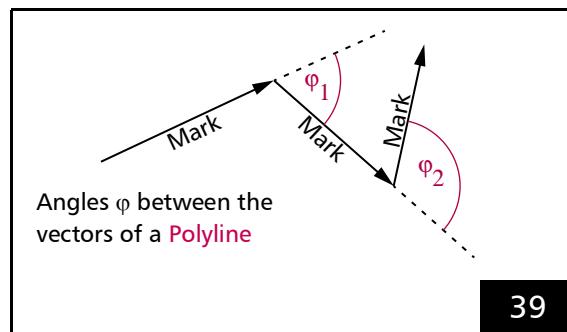
By setting the `VarPoly` value > 0 (at `set_delay_mode` or `set_delay_mode_list`) it is enabled:

- “Variable Polygon Delays” mode

Then, the RTC6 PCIe Board calculates the “Variable Polygon Delay” $v_delay(\varphi)$ for every `Polyline` corner according to:

$$v_delay(\varphi) = scale(\varphi) \times polygon_delay$$

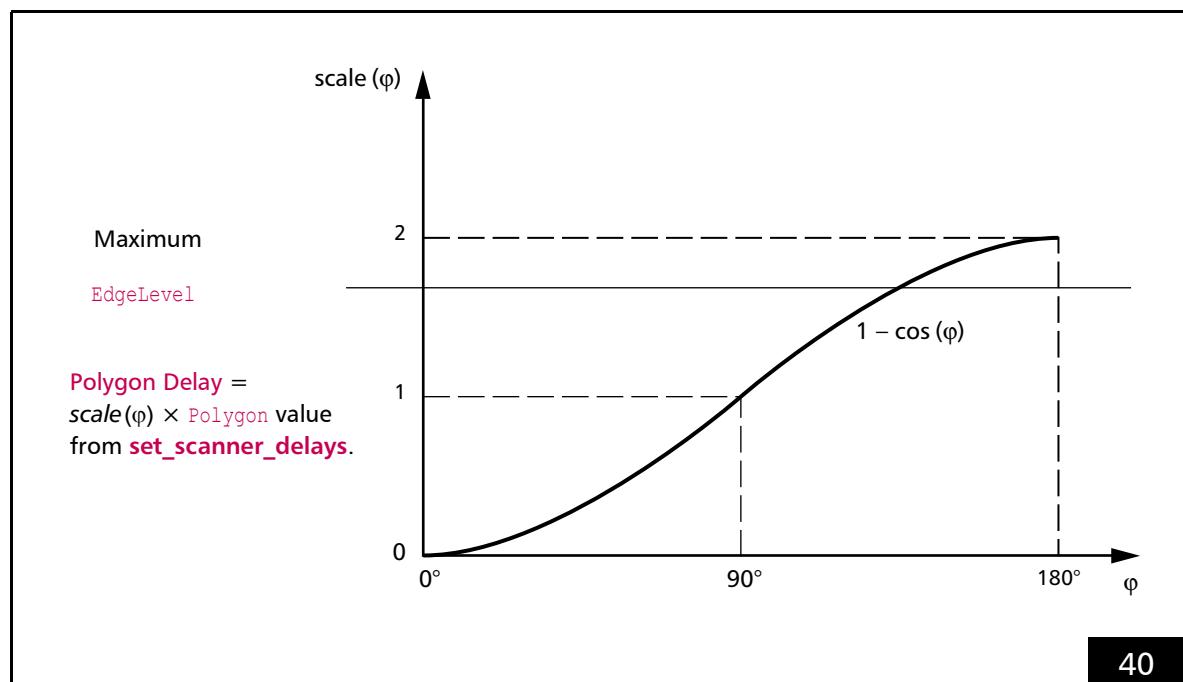
For the definition of the angle φ using the example of mark vectors, see [Figure 39](#).



39

[Variable Polygon Delays](#). Definition of the angle φ .

For circular arcs and ellipses, analogously the tangents at the connection point are relevant. `scale(\varphi)` is a scaling function for the `Polygon` value from `set_scanner_delays` (constraint $0 \leq scale(\varphi) \leq 2$), see [Figure 40](#). The default scaling function after `load_program_file` is $scale(\varphi) = 1 - \cos(\varphi)$. It can be replaced by a user-defined scaling function, see [Section “User-defined “Variable Polygon Delays””, page 152](#).



40

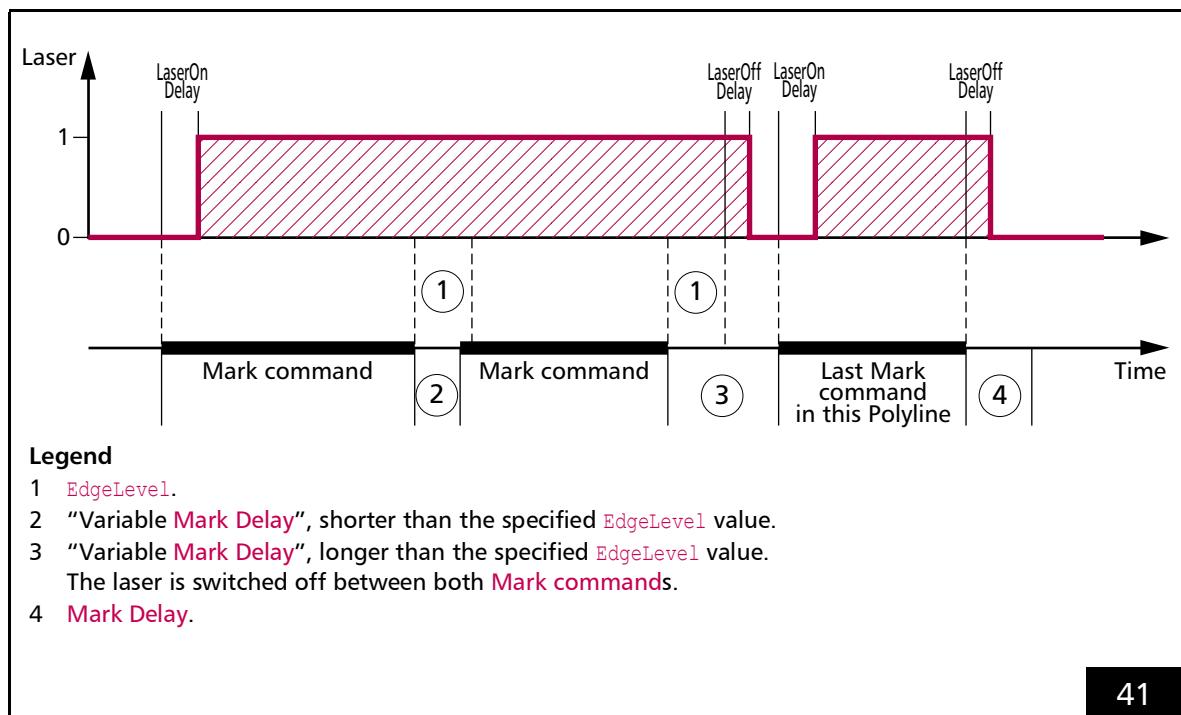
[Variable Polygon Delays](#). Variation of the `Polygon Delay` (default scaling function after `load_program_file`).

EdgeLevel

The “Variable Polygon Delay” becomes quite long with angles φ close to 180° , see Figure 40. This might lead to burn-in effects in sharp corners of a **Polyline**.

If an `EdgeLevel` value > 0 is specified (at `set_delay_mode`), the RTC6 PCIe Board switches off the laser with a **LaserOff Delay** after `EdgeLevel` delay clock cycles at the latest and thus terminates the current **Polyline**, see Figure 41.

The next **Mark command** starts as usual with the current “Variable Polygon Delay”.



41

Laser control timing during a **Polyline** with “Variable Polygon Delay”. An `EdgeLevel` value has been defined with `set_delay_mode`.

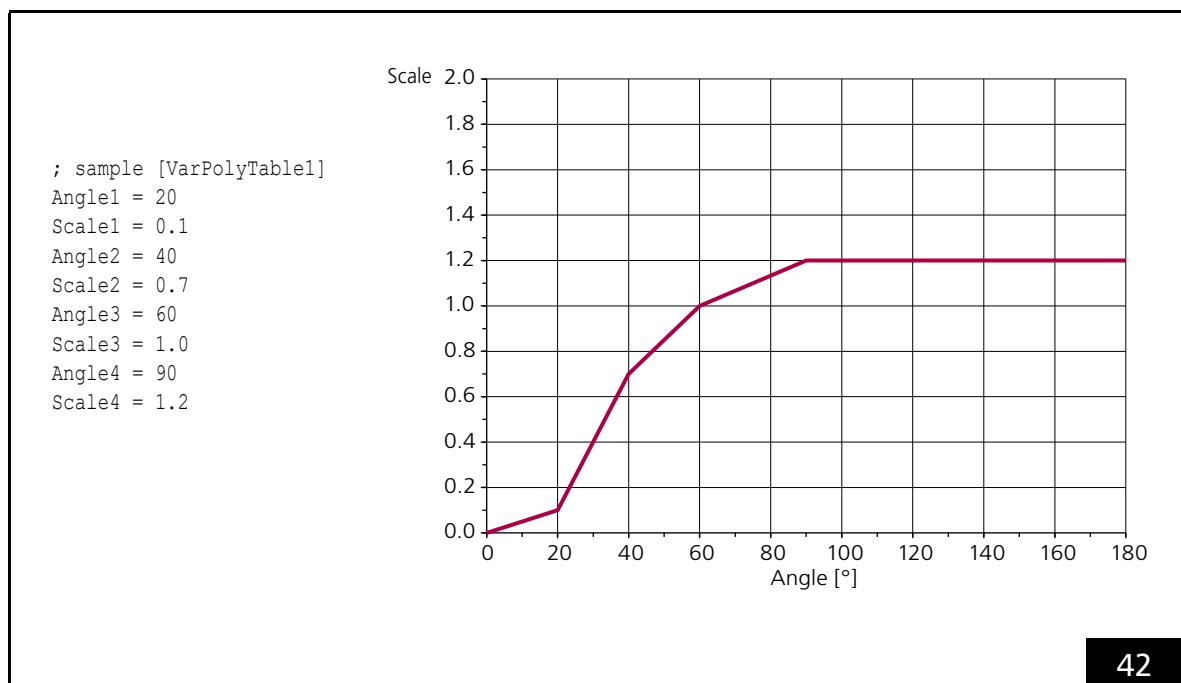
User-defined "Variable Polygon Delays"

- For the $scale(\varphi)$ scaling function, **load_varpolydelay** loads a table from an ASCII text file. See also [Section "Variable Polygon Delays", page 150](#).
- The ASCII text file can contain one or more tables⁽¹⁾.
- Each table can contain up to 50 data points ($\varphi \mid scale(\varphi)$).
- The $scale(\varphi)$ function is linearly interpolated from the data points.
- As an example, [Figure 42](#) shows a table with 4 data points and the corresponding scaling function.

Table 1: Possible table types in the ASCII text file. Each type can occur more than once.

[VarPolyTable<No>]	User-defined "Variable Polygon Delays", see page 152
[PositionCtrlTable<No>]	Scaling function , see page 198
[AutoLaserCtrlTable<No>]	Nonlinearity curve, see page 203
[JumpTable<No>]	Jump Delay values, see page 218
[StretchTable<No>]	2D stretch correction table, see page 239
[Fly2DTable<No>]	2D compensation table, see page 249

(1) Even of another type, see [table 1, page 152](#).



Example: Table for [User-defined "Variable Polygon Delays"](#) with 4 data points (left) and corresponding scaling function $scale(\varphi)$ (right).

For the tables, the following rules apply:

- Each table must begin with the line:
`[VarPolyTable<No>]`
`<No>` represents the table number.
- If the table contains multiple
`[VarPolyTable<No>]` entries with the same `<No>`,
then only the lines after the first entry are used.
Only lines up to the next '[' character (that is not
preceded by a semicolon) are used.
- Each data point (φ | $scale(\varphi)$) is defined as follows:
 $Angle< n > = < Value >$
 $Scale< n > = < Value >$
where `<n>` must be replaced by a number
 $(1 \leq < n > \leq 50)$ which denotes the number of the
data point. The values `<Value>` for the angle φ (in
degrees) and the scaling factor can be specified
as (unsigned) floating point numbers. Decimal
separator: period (.)
- 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:
 $0.0^\circ \leq \varphi \leq 180.0^\circ$ and $0.0 \leq scale(\varphi) \leq 2.0$.
- Each instruction must be in a separate line.
- Spaces and tabs in a line (for example, 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.
- 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
`load_varpolydelay` has no effect (return value 1
or 13).
- The angle $\varphi = 0^\circ$ means that two successive
vectors are parallel and are marked in the same
direction.
If the table contains no explicit data for $\varphi = 0^\circ$
(equality is checked to within $\pm 0.01^\circ$), then a
data point for $\varphi = 0^\circ$ with the scaling factor
 $scale(0^\circ) = 0$ is added.
- The angle $\varphi = 180^\circ$ means that two successive
vectors are marked in opposite directions.
If the table contains no explicit data for $\varphi = 180^\circ$
(equality is checked to within $\pm 0.01^\circ$), then a
data point for $\varphi = 180^\circ$ with the largest
scaling factor found in the table for $scale(180^\circ)$ is
added.

After initialization by `load_program_file`, the
RTC6 PCIe Board uses the internal (default) table for
the "Variable Polygon Delay" ($1 - \cos(\varphi)$, see
Figure 40). Alternatively, this can also be achieved
with `Name = NULL` in `load_varpolydelay`.

The table can be saved by `create_dat_file`.

7.2.3 Notes on Optimizing the Delays

The delays are set by `set_scanner_delays` and `set_laser_delays`.

They should be appropriate for:

- the set jump speed
- the set mark speed
- the **Tracking error** of the used scan head

If the delays are not optimized, the quality of the scanning results are under some circumstances reduced and scanning time is extended. The figures in **Section "Potential Errors when Optimizing the Delays", page 156** show the various effects of non-optimized delays on the lettering "RTC".

Notes

- **Laser Delays** are specified in units of $1/64 \mu\text{s}$.
- In contrast, the **Scanner Delays (Jump Delay, Mark Delay, Polygon Delay)** are specified in units of $10 \mu\text{s}$.

Recommended Sequence

Laser Delays should be optimized first. It is recommended to set **Jump Delays** and **Mark Delays** to a high value. The laser delay lengths have no influence on the total scan time as long as positive values are specified (see also following section).

After the **Laser Delays** have been set, the **Scanner Delays** can be optimized.

Automatic Delay Adjustments

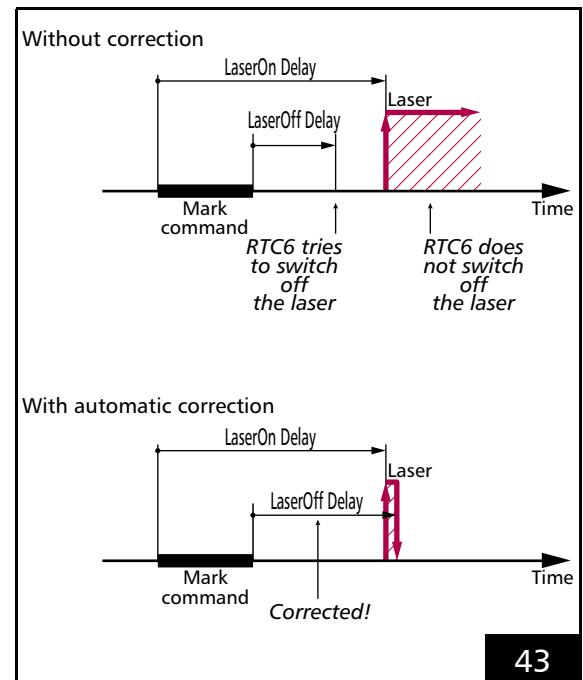
In principle, all delays can be set for any value within the corresponding allowed range. In some situations the laser control could be disturbed, for example, when Laser-On and Laser-Off overlap. Therefore, for each command that switches the laser, the RTC6 PCIe Board checks the set **Laser Delays**, corrects them and, if necessary, inserts a scanner delay "artificially". These situations, which are listed below, should actually already be avoided in the user program, because the "corrected" marking does not necessarily meet the requirements.

Laser-On and Laser-Off Overlap

The RTC6 PCIe Board corrects the faulty laser delay so that:

- the Laser-On occurs $1/64 \mu\text{s}$ after the currently effective **LaserOff Delay**
- or the Laser-Off occurs $1/64 \mu\text{s}$ after the still running **LaserOn Delay**

In both cases, the chronological next mark is cut off somewhat, see also **Figure 43**.



Automatic adjustment of **LaserOff Delay**.

43

Notes

- For further information on the general avoidance of such automatic adjustments in **RTC5 Compatibility Mode**, see **RTC5 Manual, Chapter 7.2.3 "Notes on Optimizing the Delays"**.

Laser-On and Laser-On Overlap

If the **LaserOn Delay** is switched from a long delay to a short delay between two markings, the short **LaserOn Delay** could switch on the laser before the long **LaserOn Delay** of the previous marking has expired, if the laser has been switched off between the markers (otherwise the laser would stay on and the new **LaserOn Delay** would not be effective at this point).

To avoid this overlap, the RTC6 PCIe Board in this case (like the RTC5) inserts a scanner delay “artificially”. This increases the processing time and changes the dynamics of the galvanometer scanner movement.

Laser-Off and Laser-Off Overlap

If the **LaserOff Delay** is switched from a long delay to a short delay, the short **LaserOff Delay** could switch off the laser before the long **LaserOff Delay** of the previous marking has expired, if a marking had switched on the laser in the meantime (otherwise the laser would stay off and the new **LaserOff Delay** would not be effective at this point)

To avoid this overlap, the RTC6 PCIe Board in this case (like the RTC5) extends the short **LaserOff Delay** so long that it only expires after the previous one (and of course only after the intermediate **LaserOn Delay**).

Several Simultaneously Expiring **Laser Delays**

This can happen if a laser delay is only effective beyond the length of the next command. The RTC6 PCIe Board can process up to 256 **LaserOn Delays** and **LaserOff Delays** simultaneously in a pipeline, as long as Laser-On and Laser-Off alternate and Laser-On—Laser-On and Laser-Off—Laser-Off run one after the other (see automatic adjustments above). This allows markings (interrupted lines) to be “pre-programmed” up to at least 2.56 ms.

Notes

- In contrast to the RTC6 PCIe Board, the RTC5 can only process one delay of the same type at a time. The delays are adjusted automatically. If necessary, scanner delay are inserted “artificially”. In **DSP mode < 3** (see **set_dsp_mode**), the RTC6 PCIe Board simulates the behavior of the RTC5 in order to display the same time sequences. For further details, see RTC5 Manual, Chapter 7.2.3.

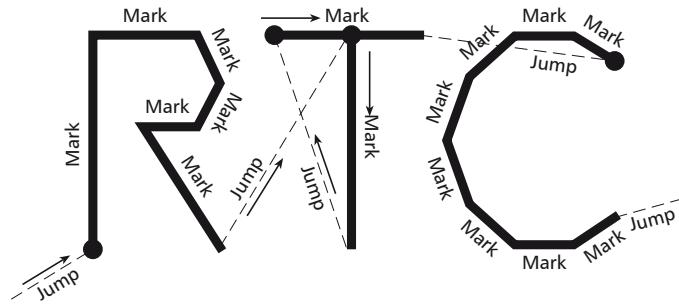
Potential Errors when Optimizing the Delays

The following figures show the various effects of unsuitable delays on the marking result, in this example 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.

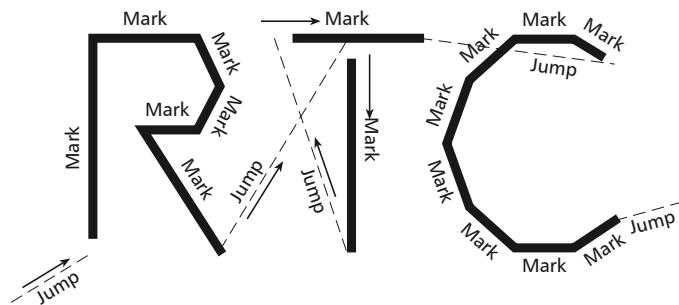
Burn-in effects at the start points of the respective vectors result.



LaserOn Delay too long

The laser is switched on too late at the beginning of a mark vector.

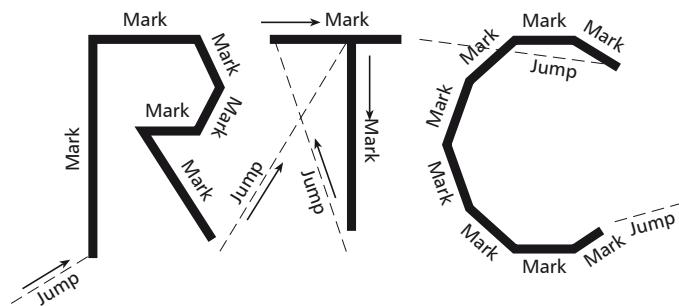
The first part of the vector is not marked.



LaserOff Delay too short

The laser is switched off after a mark command before the mirrors have reached the vector end position.

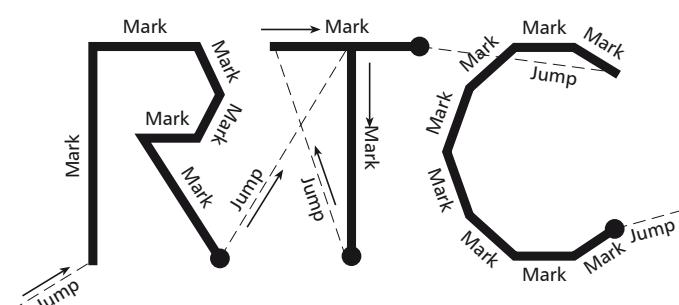
The respective vectors are not marked completely.



LaserOff Delay too long

The laser is switched off too late after a Mark command. The laser is still on, although the mirrors are already slowed down considerably or have already stopped.

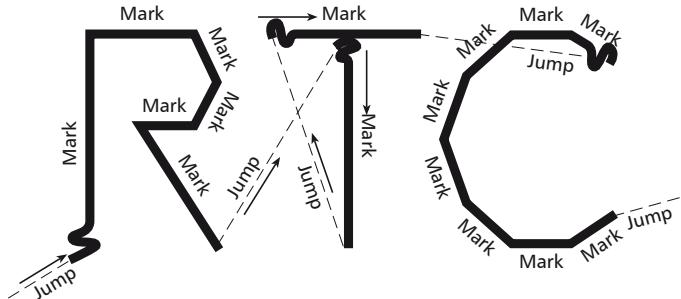
The results are burn-in effects at the end points of the respective vectors.



Jump Delay too short

The mark vector that follows a **Jump command** has already started although the scanners have not yet settled.

A running-in oscillation or overshoot is visible.



Jump Delay too long

There are no visible effects if the **Jump Delay** is too long.

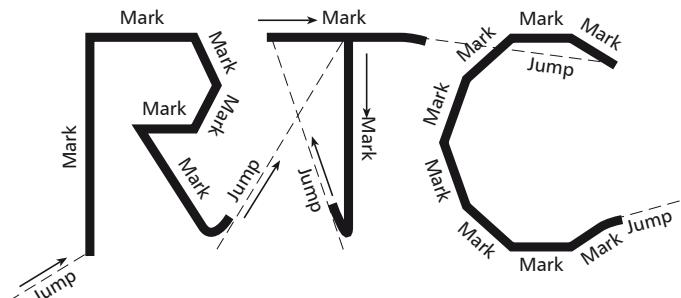
However, the scanning time is extended.



Mark Delay too short

The traversing of a vector is started before the mirrors have reached the end position of the preceding mark vector.

The end of the mark vector is turned towards the direction of the jump vector.



Mark Delay too long

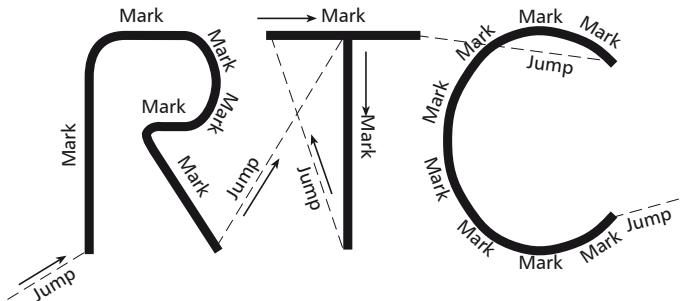
There are no visible effects if the **Mark Delay** is too long, but the scanning time is increased.



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.

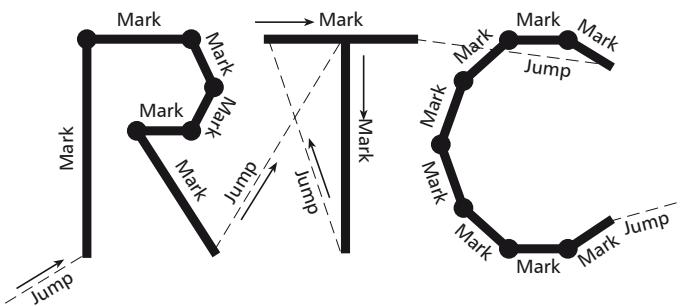
The corners of the **Polyline** are rounded off.



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

"Variable Polygon Delays" mode

, a maximum length
("EdgeLevel") can be defined,
see **Section "EdgeLevel"**,
page 151 for details.

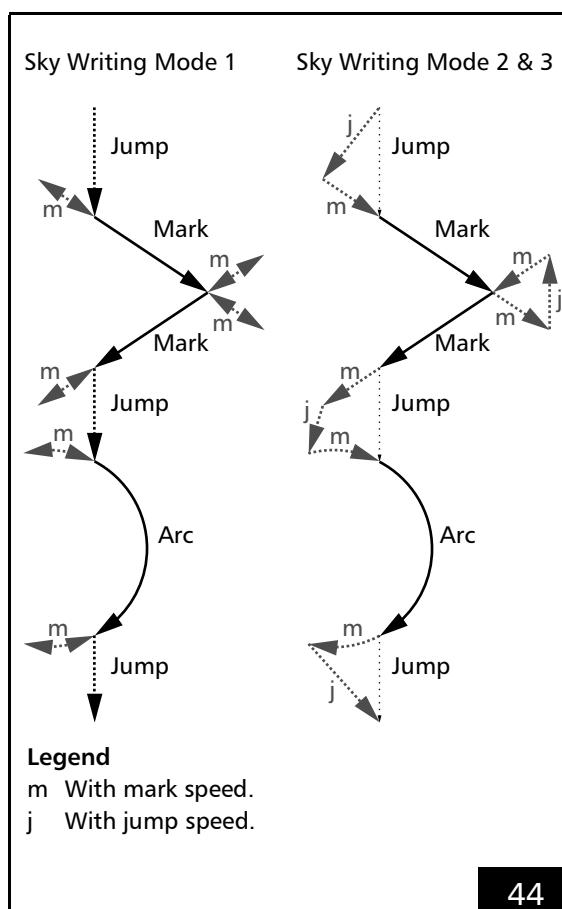
7.2.4 Sky Writing

For applications with elevated accuracy requirements the so-called **Sky Writing** can be switched on. Then, every mark vector is precisely executed at a constant mark speed over the entire vector length.

For **Mark commands** with **Sky Writing** enabled, the RTC6 PCIe Board automatically performs non-marking **Sky Writing** scan motions before and after the to-be-executed vectors and arcs, see [Figure 44](#).

The following are always executed without **Sky Writing**:

- **micro_vector[*] Commands**
- **[*]para[*] Commands**



Sky Writing motions (grey). For **Sky Writing Mode 3** see also [Figure 45](#).

By the **Sky Writing** command parameters you can specify:

- the duration of these run-in motions
- the duration of these run-out motions
- the synchronization between scan motion and laser control

Sky Writing is available in the following modes:

- [Sky Writing Mode 1, page 159](#)
- [Sky Writing Mode 2, page 160](#)
- [Sky Writing Mode 3, page 161](#)

Sky Writing Mode 1

In **Sky Writing Mode 1**, the RTC6 PCIe Board performs the following **Sky Writing** motions for each **Mark command** – regardless of previous or subsequent commands:

- In the run-in phase, the vector/arc is preceded by a “forerun” movement performed by the galvanometer scanners at mark speed, see [Figure 44](#): the galvanometer 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/arc has been processed at mark speed, it gets a short deceleration and retrace movement of the galvanometer scanners (at mark speed) appended in the run-out phase.

Sky Writing Mode 1 can be switched on/off by:

- [set_sky_writing](#)
- [set_sky_writing_list](#)
- [set_sky_writing_para](#)
- [set_sky_writing_para_list](#)

Sky Writing Mode 2

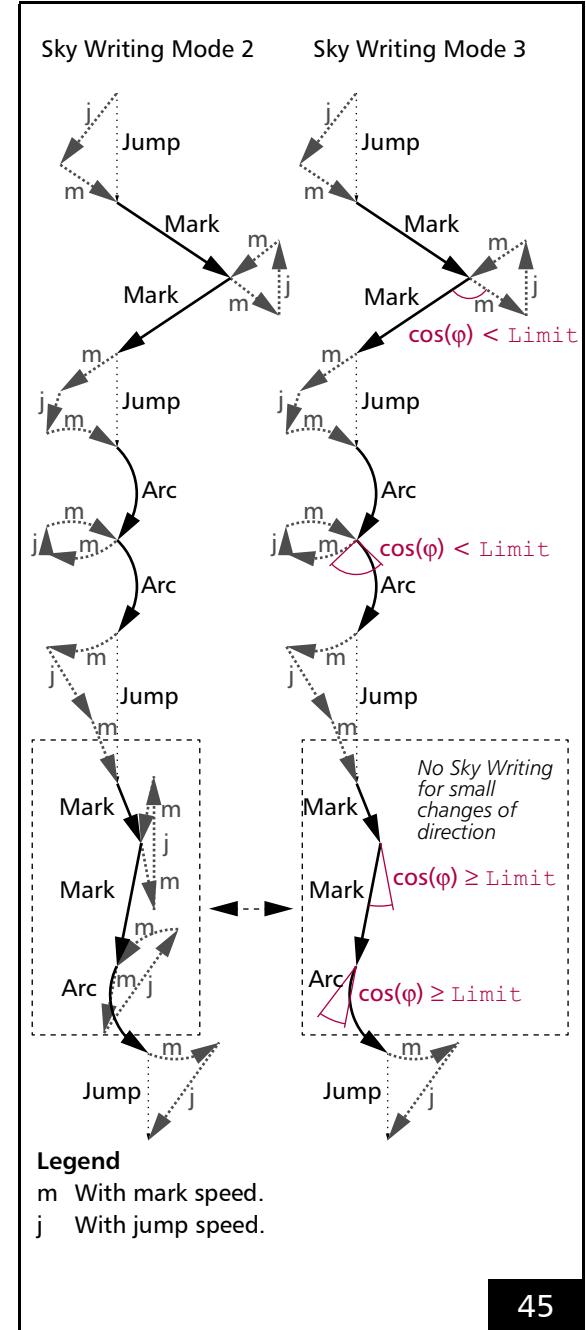
In **Sky Writing Mode 2**, the RTC6 PCIe Board calculates *time-shortened* **Sky Writing** motions.

Here, too, each to-be-executed vector/arc gets preceded and appended with a run-in motion and a run-out motion in extension of the vector/arc at mark speed. Within a **Sky Writing Mode 2** marking sequence, however, neither forerun motions (in the run-in phase) nor retrace motions (in the run-out phase) of the galvanometer scanners occur. Instead, the RTC6 PCIe Board executes **Sky Writing** jumps (at the currently specified 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 44](#).

`set_sky_writing` and `set_sky_writing_para` always switch on **Sky Writing Mode 1**. Therefore, it is only possible to *change the mode afterwards to Sky Writing Mode 2 by `set_sky_writing_mode` or `set_sky_writing_mode_list`*.

Sky Writing Mode 2 activation affects the same commands as **Sky Writing Mode 1** (except ellipse commands, see below). *Time-shortened Sky Writing*, however, can only occur within a sequence of non-parameterized `[*]mark[*]` Commands, arc commands and **Jump commands** (may also contain timed or 3D commands). If such a sequence gets interrupted by some other “non-**Sky Writing Mode 2-capable**” list command (for example, a `[*]para[*]` Command, ellipse command or any short list command), then the RTC6 PCIe Board suspends **Sky Writing Mode 2** and complete the preceding command in **Sky Writing Mode 1** (with a retrace motion of the galvanometer scanners). This suspension of **Sky Writing Mode 2** does not deactivate it: subsequent “**Sky Writing Mode 2-capable**” commands are started at in **Sky Writing Mode 1** (with a forerun motion of the scan head) and then executed again in **Sky Writing Mode 2**.

Even with **Sky Writing Mode 2** switched on, ellipse commands are always executed in **Sky Writing Mode 1**.



Sky Writing 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.

Therefore, `set_sky_writing_mode` or `set_sky_writing_mode_list` can be used to switch to **Sky Writing Mode 3**. A switching limit angle can be defined for this by `set_sky_writing_limit` or `set_sky_writing_limit_list`.

Then only for larger angle deviations ($\cos(\varphi) < \text{Limit}$) between successive **Mark commands** of a **Polyline** a **Sky Writing** motions is executed as in **Sky Writing Mode 2**.

In contrast, smaller angular changes ($\cos(\varphi) \geq \text{Limit}$) result in a (variable) polygonal delay, see **Section "Variable Polygon Delays", page 150**. See also **Figure 45**.

Notes

- In case a **Polyline** ends with a short mark vector (shorter than mark speed \times 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**.
- At the beginning and end of a **Polyline**, as well as when interrupted with "non-**Sky Writing Mode 2**-capable" list commands, a **Sky Writing Mode 1** movement always occurs in **Sky Writing Mode 3**, see **Section "Sky Writing Mode 2", page 160**.

Synchronization

The timing diagram for scan-head and laser control in **Sky Writing Mode 1** is shown in **Figure 46**. The timing diagram for **Sky Writing Mode 2** and **Sky Writing Mode 3** is similar, but here the galvanometer scanners perform no reverse motion in the run-in and run-out phases.

The **Timelag**, **Nprev**, **Npost** and **LaserOnShift** parameters are specifiable for `set_sky_writing_para` and `set_sky_writing` and the corresponding list commands:

- The **Nprev** parameter is a whole number in units of $10 \mu\text{s}$ and defines the duration of the run-in:

Mode	Duration
1	$20 \times \text{Nprev} [\mu\text{s}]$
2, 3	$10 \times \text{Nprev} [\mu\text{s}]$

- The **Npost** parameter is a whole number in units of $10 \mu\text{s}$ and defines the duration of the run-out:

Mode	Duration
1	$20 \times \text{Npost} [\mu\text{s}]$
2, 3	$10 \times \text{Npost} [\mu\text{s}]$

- The parameters **Timelag** (64-bit IEEE floating point value rounded to integer multiple of $1/64 \mu\text{s}$) and **LaserOnShift** (whole number in units of $1/64 \mu\text{s}$) define the delay of the **Signals for "Laser Active" Operation** 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 + $1/64 \mu\text{s} \times \text{LaserOnShift}$
 - Delay of switch-off time point relative to the set ending position / μs
= Timelag



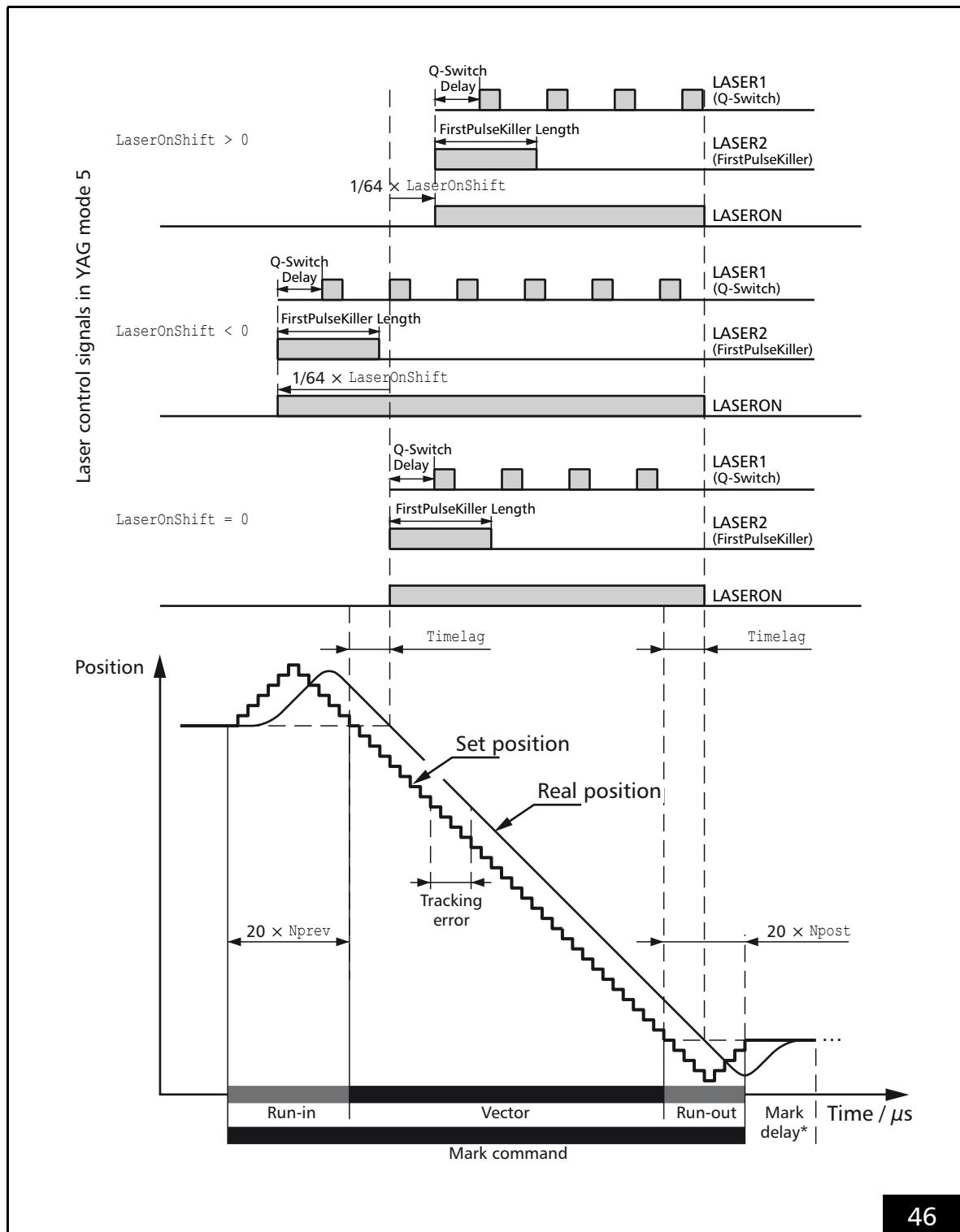
For `LaserOnShift = 0`, the **Signals for "Laser Active" Operation** are switched on (off) with a delay of `Timelag` relative to the set starting position (set ending position).

By setting the parameter `Timelag` to the actual **Tracking error** (and the parameters `Nprev` and `Npost` to sufficiently large values), it is ensured that:

- the **Signals for "Laser Active" Operation** switch on/off precisely at the endpoints of the desired vector
- the **Tracking error** becomes constant even before the vector's startpoint is reached
- the vector is executed right up to its endpoint with constant mark speed

The **Laser Delays** from `set_laser_delays` are not effective with **Sky Writing**.

The switch-on time point of the **Signals for "Laser Active" Operation** can be adjusted with the parameter `LaserOnShift` (for example, to the `HalfPeriod` of `set_laser_pulses` or to the reaction times of the laser itself), so that the first laser pulse occurs exactly with the set starting position. Positive `LaserOnShift` values delay the switching on of the laser control signals. Negative `LaserOnShift` values advance the switching on of the laser control signals (at the earliest, however, until the beginning of the forward run). Negative values are also necessary to "make room" for a Q-Switch delay or a `FirstPulseKiller` signal in one of the YAG modes.



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)
 (* This **Mark Delay** is only effective with **Sky Writing Mode 3** or **Sky Writing Mode 0**).

Notes

- By `set_sky_writing` and `set_sky_writing_mode_list`, you can switch back and forth between **Sky Writing Mode 1** and **Sky Writing Mode 2** or **Sky Writing Mode 3**. Temporarily switching off is also possible, as long as `Timelag > 0`.
- When searching for appropriate **Sky Writing** parameters, initially `set_sky_writing` and **Sky Writing Mode 1** should be used: as start value for the `Timelag` parameter, the **Tracking error** of the galvanometer scanners should be set. Then `Timelag` (with `LaserOnShift = 0`) can be fine-tuned such that marking vector *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 `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`.
- With `set_sky_writing` and `set_sky_writing_list`, `Nprev` and `Npost` are predefined:
 - `Nprev = approx. 0.15 × Timelag`
 - `Npost = approx. 0.1 × Timelag`
This results in the following run-in and run-out durations:

	Mode	Duration [Timelag]
Run-in	1	3
Run-in	2, 3	1.5
Run-out	1	2
Run-out	2, 3	1

- With `set_sky_writing_para` or `set_sky_writing_para_list`, you can also specify other run-in and run-out phases, for example, shorter ones to reduce marking times (but this may be at the expense of accuracy). Shorter run-in and run-out phases are particularly beneficial when lines are marked in only one direction without interruption and jump speed and mark speed are set equally. Note that in **Sky Writing** mode no automatic adjustment of laser and **Scanner Delays** is performed:
 - If the next **Mark command**'s run-in phase begins when the preceding **Mark command**'s **LaserOn Delay** has not yet expired, then the laser does not switch on for the preceding command.
 - If the next **Mark command**'s run-out phase begins when the preceding **Mark command**'s **LaserOff Delay** has not yet expired, then the laser does not switch off for the preceding command.
 - In **Sky Writing Mode 1**, a positive `LaserOnShift` time (in μ s) should exceed the smallest occurring marking time by no more than $(20 \times Npost - Timelag)$ – and in **Sky Writing Mode 2** and **Sky Writing Mode 3** by no more than $(10 \times Npost - Timelag)$. Otherwise, the laser is not switched on for this marking.

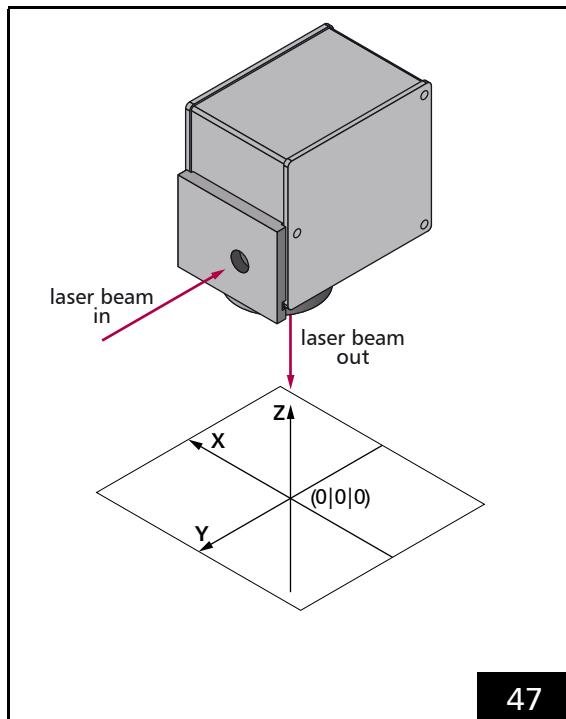
- Pulse lengths and frequencies (in YAG modes additionally the Q-Switch delay and the FirstPulseKiller signal parameters) previously defined for the **Signals for "Laser Active"** **Operation** are kept unchanged in the **Sky Writing** mode. On the other hand, previously defined LaserOn, LaserOff, mark, 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 addition of a **Mark Delay** defined prior to activation (see [Figure 46](#)).
- In **Sky Writing Mode 1**, the run-in and run-out phases take place fully within the respective **Mark command**. As a result, execution of the **Mark command** is extended by a time period (in 10 μ s) of $(2 \times N_{prev} + 2 \times N_{post})$. In **Sky Writing Mode 2** and **Sky Writing Mode 3** execution is extended by a time period (in 10 μ s) of at least $(N_{prev} + N_{post})$.
- In **Sky Writing Mode 1**, **Jump Delays** are performed normally. The **Jump Delay** value (in 10 μ s, see [set_scanner_delays](#)) can be reduced by approx. $(2 \times N_{prev})$ or even to 0. In **Sky Writing Mode 2** and **Sky Writing Mode 3**, the **Jump Delay** is automatically (internally, dynamically) reduced by up to N_{prev} .
- Time-based **[*]mark[*] Commands** and **"Arc" commands** can also be performed in **Sky Writing** mode, see [Chapter 8.9 "Timed Commands", page 273](#). Here, however, a shorter marking period is coupled with a higher mark 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 mark speed.
- the **Sky Writing** mode is not taken into account (but also not deactivated)
 - During execution of **[*]para[*] Commands**, for example, with activated "vector-controlled laser control", see [Section "Vector-Defined Laser Control", page 205](#)
 - During execution of **micro_vector[*] commands**, see [Chapter 8.8 "micro_vector\[*\] Commands", page 272](#)
- With **Sky Writing Mode 2** or **Sky Writing Mode 3** activated, the following functionalities of the 2D **Jump commands** **jump_abs** and **jump_rel** are not executed:
 - Automatic tuning switching in **Jump Mode**, see [Chapter 8.1.5 "Jump Mode", page 216](#)
 - Coordinate transformations with **at_once = 2**, see list item "With **at_once = 2 ...**", [page 225](#).

7.3 Scan Head Control

7.3.1 Reference System

The reference system for the **Image field** which is used by the RTC6 PCIe Board is shown in [Figure 47](#).

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, that is, the point $(0|0|0)$, is in the center of the **Image field**.



Reference system for the RTC6 coordinates.

7.3.2 Image Field Size and Image Field 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 (that is, the distance between the input laser beam axis and the **Image field**).

The x, y and z coordinates of a vector must be specified as signed 20-bit values (that is, as numbers between $-524,288$ and $+524,287$).

The calibration factor K defines the ratio of the digital point coordinates in *bits*⁽¹⁾ and the actual position of the point in *millimeters*.

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⁽¹⁾].

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 a used correction table by [get_table_para](#) or [get_head_para](#).

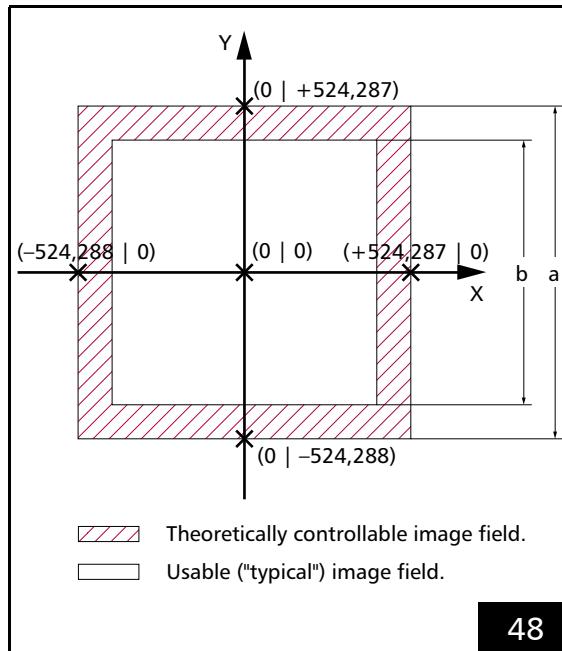
Given the calibration factor K , the side length a of the usable **Image field** in *millimeters* can be calculated:

$$a = \frac{2^{20}}{K}$$

(1) The expression "bits" is here synonymous with "digital control value" (see footnote ⁽³⁾ on [page 135](#)).

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**.



Compatibility Modes

RTC6 Standard Mode

(Default after `load_program_file` and `set_RTC6_mode`)

The **Image field** coordinates for the x axis, y axis and z axis and all related parameters (for example, jump speed or wobble amplitude) are to be specified as signed 20-bit values.

RTC5 Compatibility Mode

(`set_RTC5_mode`)

The **Image field** coordinates for the x axis and y axis and all associated parameters are to be specified as signed 20-bit values – as in **RTC6 Standard Mode**. See also [Chapter 2.12.2 “Adapting RTC5 Source Code for the RTC6 PCIe Board”, page 59](#).

The **Image field** coordinates of the z axis are to be specified as signed 16-bit values. The **RTC6 PCIe Board** automatically multiplies all z coordinates by 16⁽¹⁾.

As the z calibration factor, a value 16 times smaller must be used.

RTC4 Compatibility Mode

(`set_RTC4_mode`)

The **Image field** coordinates for the x axis, y axis and z axis and all related parameters are to be specified as signed 16-bit values. The **RTC6 PCIe Board** automatically multiplies them by 16⁽¹⁾.

As the xy calibration factor and z calibration factor, a value 16 times smaller must be used. See also [Chapter 2.10.2 “Porting RTC4 Source Code to the RTC6 PCIe Board”, page 48](#).

Image field size.

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 48](#)).
- If the laser focus is to 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**:

$$M = K \times b/2$$

(1) The allowed value range decreases accordingly.

7.3.3 Virtual Image Field

In particular for Processing-on-the-fly applications, a virtual **Image field** of size 29-bit⁽¹⁾ is available.

Therefore, **Vector commands** and **“Arc” commands** can be loaded for objects up to 512 times larger than the real **Image field** (in the Processing-on-the-fly direction). For details on Processing-on-the-fly in the virtual **Image field**, see [Chapter 8.6.6 “Virtual Image Field with Processing-on-the-fly”, page 251](#).

At runtime, the current coordinates are clipped to the real **Image field** [-524,288...+524,287], see [2...7](#) in [Chapter 7.3.6 “Output Values to the Scan System”, page 180](#).

In addition, the extended value range of the virtual **Image field** can also be used for utilizing the complete real 20-bit **Image field** during coordinate transformations (such as rotations, shrinkages or shifts, see [Chapter 8.2 “Coordinate Transformations”, page 223](#)). Otherwise, certain edge areas of the real **Image field** may remain inaccessible during such coordinate transformations.

Coordinate Transformations in the Virtual Image Field

“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**.

See [3](#) in [Chapter 7.3.6 “Output Values to the Scan System”, page 180](#).

“Virtual coordinate transformations” are defined by:

- `set_matrix(HeadNo = 4)`
- `set_offset(HeadNo = 4)`
- `set_offset_xyz(HeadNo = 4)`
- `set_angle(HeadNo = 4)`

They let you define matrix coefficients and 2D offsets (with `set_offset_xyz`, `zOffset` is ignored). With `at_once = 1`, they become effective immediately, if no list is currently being processed. Otherwise, they are only saved as with `at_once = 0`.

Stored transformations are automatically activated when a **Processing-on-the-fly session** is restarted (the changed output position is reached at jump speed). They remain active⁽²⁾ even after the end of this **Processing-on-the-fly session**.

As long as a **Processing-on-the-fly session** is active, the parameters for the “virtual coordinate transformation” cannot be changed, they can only be saved.

By `set_matrix(4, 0.0, 0.0, 0.0, 0.0)`, the “virtual coordinate transformation” can be deactivated again, see [Section “Clock Overruns”, page 181](#).

The adjustment range for offsets is ± 28 bits. Matrix coefficients must not exceed an absolute value of 2.0. The offset is applied after the matrix operation.

For “virtual coordinate transformations” *cannot* be used:

- `set_scale`
- all list commands for coordinate transformations

With RTC6 Software Package \geq V1.6.1, “virtual coordinate transformations” can be also defined by a **“Global Online Positioning”**, see [Chapter 8.3.2 ““Global Online Positioning””, page 230](#).

(1) Up to DLL 608: 24-bit.

(2) With RTC6 Software Package $<$ V1.5.0, “virtual coordinate transformations” are only available during a `set_fly_2d` **Processing-on-the-fly session**.

7.3.4 3D Image Field

Carrying Out Adjustment

SCANLAB 3D correction tables are calculated such that the plane of the middle focus position is controlled with $z = 0$.

Therefore, the mechanical distance between scan system and working plane must be adjusted accordingly. With the varioSCAN series, a specific distance to the scan system has to be maintained in addition. The values are to be taken from the manual of the respective 3-axis scan system or varioSCAN.

Deviations from these should preferably be set via the user program by `set_defocus`, `set_defocus_offset`, `set_offset_xyz` or the corresponding list commands.

Once the working distance and distance to the scan system are correctly adjusted, then subsequently the laser focus position should be fine-tuned. For this, a test pattern is to be marked in the middle of the $z = 0$ working plane. The optimum laser focus position for processing results can be achieved:

- With the varioSCAN series, by manually turning the focusing ring, see corresponding Manual
- With a varioSCAN FLEX, by adjusting the focusing optic position, see Manual for "RTC5/6 varioSCAN 40 FLEX Extension" extension board (#128683)
- With a varioSCAN FC and intelliWELD FC by adjusting the coefficient A of the used 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 information is generally not available to SCANLAB and therefore, are not included in the calculation of 3D correction tables.

Therefore, in some cases the calculated 3D correction table might not fit optimally the individual scan system. To test whether this is the case, run a laser marking test that covers the entire **3D image field**.

Check if the laser focus meets the requirements of your application. If you find that the spot diameter varies considerably, then a recalibration of the z axis correction may help under certain circumstances.

The aim of the z axis calibration procedure is to determine suitable coefficients A, B and C. These can be transferred to the RTC6 PCIe Board by `load_z_table`.

A, B and C are coefficients of the parabolic function

$$z_{\text{out}} = A + BI + CI^2$$

(focus length value I in the RTC4 compatibility range $[-32,768 \dots +32,767]$, see `load_z_table_no`; See also [Notes](#) below).

They determine the relationship between the z output value z_{out} and the focus length value I . For each point $(x|y|z)$ in the **3D image field**, the focus length value I corresponds to the focus length difference between the specified point $(x|y|z)$ and the point $(0|0|0)$.

The ABC coefficient values of a correction file on the PC can be read-out directly with `read_abc_from_file` and written into with `write_abc_to_file`, see following [Notes](#).

(1) Coefficients A, B and C can be queried by `get_head_para` and newly set by `load_z_table`. Only A should be modified. B and C should be passed over unchanged – as queried – by `load_z_table`.

Notes

- As of RTC6 Software Package V1.12.0, you can leave the focus length value l in RTC6 20-bit range $[-524,288\dots+524,287]$, if you use **load_z_table_20b** / **load_z_table_no_20b** and **read_abc_from_file_20b** / **write_abc_to_file_20b**.
- ABC values from the ***_ReadMe.txt** file of correction file are to be interpreted as 16-bit values.

Procedure

- Refer also to the "Calibrating a 3-Axis Laser Scan System" Manual.
 - See also **Notes**, page 170.
- (1) Adjust the correct mechanical distance between the scan system and the $z = 0$ working plane and the correct distance between the varioSCAN device and the scan system⁽¹⁾.
 - (2) Load the 3D correction file by **load_correction_file**.
 - (3) Assign the 3D correction table by **select_cor_table**.
 - (4) Read out the assigned coefficients A , B and C by **get_head_para**.
 - (5) varioSCAN FC and intelliWELD FC only: proceed with step 10. Steps 6...9 do not need to be performed.
 - (6) Move the laser spot to the reference point⁽²⁾ by **goto_xyz**.
 - (7) Set the z axis to the neutral (middle) position by **load_z_table_20b**(0, 0, 0).
 - (8) Place a test object at the reference point.
 - (9) Adjust the laser focus, see page 169.
 - (10) Move the focus to an arbitrary point $(x|y|z)$ within the (required) **3D image field** by **goto_xyz**⁽³⁾.

- (1) See the corresponding manual.
- (2) 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 should be focused to the reference point when the z axis is set to the neutral position (Z output value $z_{out} = 0$). The coordinate values of the reference point (in mm) are provided in the ***_ReadMe.txt** file that accompanies the 3D correction file or in the 3-axis scan system's or the varioSCAN user manual. If you are using an F-Theta objective, the reference point is generally the origin $(0|0|0)$.

(11)Query the focus length value l (in RTC6-20-bit range $[-524,288\dots+524,287]$) set by the RTC6 PCIe Board for this point⁽⁴⁾ by **get_z_distance**.

(12)Optimize the laser focus at this point: vary the Z output value z_{out} until the quality of the laser focus meets your requirements⁽⁵⁾ by **load_z_table_20b**($A=z_{out}$, 0, 0). A starting value can be calculated 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 .

(13)Repeat steps 10...12 for as many locations $(x|y|z)$ as possible⁽⁶⁾ and write down the values $(l|z_{out})$ for each new point. If possible, seek to thereby cover the entire **3D image field** required by your application.

(14)Fit the function $z_{out} = A + Bl + Cl^2$ to your value pairs $(l|z_{out})$.

(15)Use the resulting coefficients A , B and C to adjust the 3D correction table by **load_z_table_20b**(A , B , C).

Values loaded by **load_z_table_20b** are overwritten by a subsequent **load_correction_file**

(load_correction_file sets the three coefficients A , B and C to the values of the loaded correction table). After each **load_correction_file** or **select_cor_table**, you should therefore also call **load_z_table_20b**(A , B , C) again.

By **load_z_table_no_20b**, the three coefficients A , B , C can be assigned to correction table **No**. They are switched with **select_cor_table**.

Alternatively, the ABC values can be written permanently to the header of the correction file by **write_abc_to_file_20b**, see also parameter 5...7 in Section "ct5 Correction File Header", page 177.

- (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 **3D image field** is specified in the corresponding user manual.
- (4) The focus length value l can be positive or negative, see notes on **get_z_distance**.
- (5) The optimal z_{out} output value can be positive or negative.
- (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.



Testing 3-Axis Scan Systems with F-Theta Objective

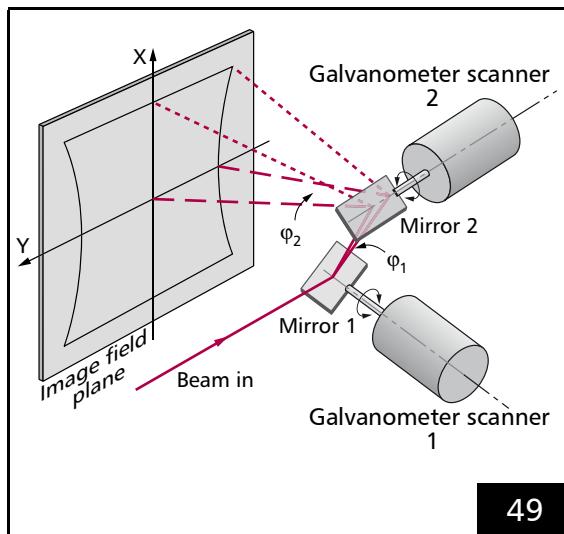
- With the adjusted 3D correction table and 3D vector commands, see [Chapter "3D Commands"](#), page 237, mark two identical large squares. Mark one of the squares with the work piece in z position $z = z_{\min}$, the other one with $z = z_{\max}$.
- These two squares should match exactly. If this is 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.
- See also [Section "Enhanced 3D Correction"](#), page 239.

7.3.5 Image field Correction and Correction Tables

Field Distortion

The deflection of a laser beam with a two-mirror system results in three effects:

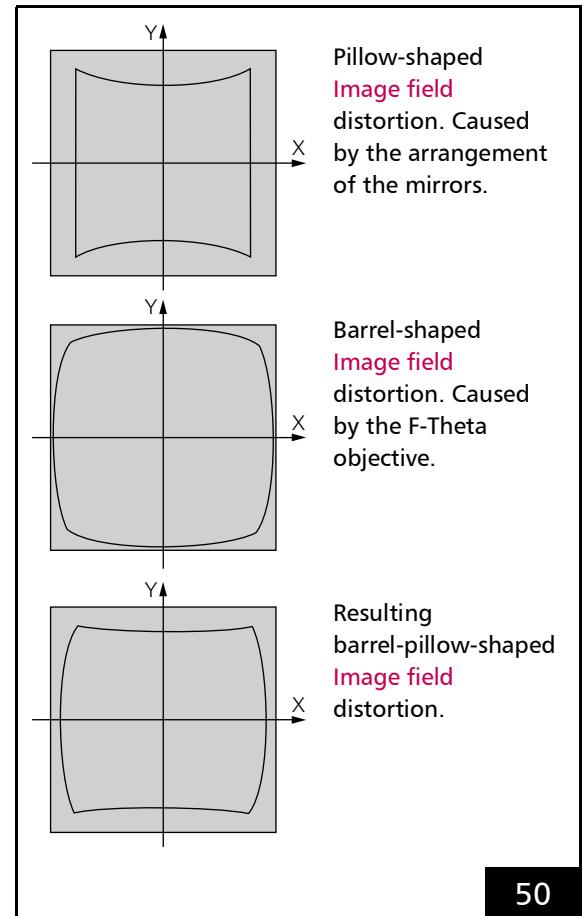
- (1) The arrangement of the mirrors leads to a certain distortion of the **Image field**⁽¹⁾, see Figure 49.
- (2) The distance in the **Image field** is not proportional to the scan angle itself, but to the tangent of the scan angle. Therefore, the mark speed of the laser focus in the **Image field** 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** plane, a varying spot size results.



49

Image field distortion when deflecting a beam in a two-mirror deflection system.

By focusing the deflected laser beam with an F-Theta objective, effect 2 and effect 3 can be avoided. However, this causes a barrel-shaped distortion of the **Image field**, see Figure 50.



50

Image field distortion caused by the arrangement of the mirrors and by the F-Theta objective.

- (1) Cause: 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.

Image Field Correction Algorithm

For these field distortions, the RTC6 PCIe Board board internally uses a algorithm to compensate for it. The algorithm is based on a correction table.

An orthogonal grid of 257×257 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 RTC6 PCIe Board calculates the corrected coordinates by interpolating from the grid points in the correction table.

The correction is executed for every single **Microstep**, see also [Chapter 7.1.2 "Microstepping", page 139](#).

SCANLAB creates a correction file for every system type. For this purpose, the correction table is calculated based solely on general system data (such as mirror geometry, calibration and the objective 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 RTC correction files based on test measurement data. For further information, refer to the corresponding manual or contact SCANLAB).

Activating Image Field Correction

To activate **Image field** correction, at the beginning of a user program the required correction tables must:

- (1) be loaded from the corresponding correction files into RTC6 PCIe Board memory (by [load_correction_file](#))
- (2) assigned to the used scan head connectors (by [select_cor_table](#) or [select_cor_table_list](#))

2D correction tables activate an **Image field** correction for x and y coordinates,
3D correction tables additionally for z coordinates.

2D and 3D Correction Files

SCANLAB supplies 2D and 3D correction files. They have the extension ***.ct5** and the following naming scheme applies:

- **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 header, see also [Section "ct5 Correction File Header", page 177](#).

Loading Correction Tables

By `load_correction_file`, up to 8 correction tables can be loaded from their corresponding correction file into the RTC6 PCIe Board memory, see also [Chapter 8.5 "Controlling 2D Scan Systems and 3D Scan Systems", page 235](#).

If the **Option "3D"** is not enabled, then only 2D correction tables can be loaded.

2D correction tables can also be loaded from 3D correction files (by `load_correction_file` with `Dim = 2`). The 3D part is ignored.

If the **Option "3D"** is enabled, a 3D correction table can even be loaded from a 2D correction file (by `load_correction_file` with `Dim = 3`). Thereby, the 2D correction table is automatically supplemented with a linear z correction.

The actually suitable Z correction can be loaded by `load_z_table_no` or `load_z_table` after the assignment by `select_cor_table` (see below), see [Chapter 7.3.4 "3D Image Field", page 169](#).

Notes

- RTC5/RTC6 correction files (file extension `*.ct5`) differ from those of prior RTC products (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 [Section "Converting Correction Files", page 179](#).

Assigning Loaded Correction Tables

Loaded correction tables are assigned to the two scan head connectors by `select_cor_table` or `select_cor_table_list`.

If neither the **Option "Second Scan Head Control"** nor the **Option "3D"** is enabled, then a 2D correction table can be assigned only to the first scan head connector.

If the **Option "Second Scan Head Control"** is enabled, then the two scan head connectors can each be assigned their own 2D correction table.

If the **Option "3D"** is enabled but not the **Option "Second Scan Head Control"**, then a 2D correction table or a 3D correction table can be assigned exclusively to the first scan head connector. The first scan head connector outputs position signals for an xy scan system.

With a 3D correction table, the second scan head connector outputs additionally position signals for the z axis (for example, varioSCAN) via both channels.

If the **Option "Second Scan Head Control"** and the **Option "3D"** are enabled, up to two (even different) 2D correction tables or a single 3D correction table can be assigned.

In order to control a 3D system, the (only) 3D correction table must be assigned exclusively to the connector for the xy scan system. There is no assignment for the z axis. The output for the z axis occurs automatically on both channels of the other port (`select_cor_table(n, 0)` or `select_cor_table(0, n)`).

Notes

- If no correction table has been loaded into the RTC6 PCIe Board memory, then the board outputs unforeseen values to the scan system. Therefore, at least a 1to1 correction table must be loaded (as 2D or 3D correction table), see **Section "1to1 Correction Tables", page 176**, if no 2D correction file or 3D correction file is available that has been calculated for the specific optical configuration.
- Correction files should have been assigned⁽¹⁾:
 - before a list is started for the first time or
 - before the galvanometer scanners of a scan system are moved by a control command (like **goto_xy**)
- If only one scan head connector has an 2D correction table assigned, then the other scan head connector outputs no *position* signals.
- During initialization, **load_program_file** assigns a correction table according to **select_cor_table(1, 0)**. However, it leaves the galvanometer scanners at position (0,0). **load_program_file** cannot determine whether a correction table #1 is loaded at all. Therefore, there is no internal jump to the output position that is specified in the correction file. Its contents could be random (see above).

- **load_program_file** does not initialize the memory contents of the correction tables. The correction table values are random immediately after switching on the RTC6 PCIe Board. It is recommended to explicitly call **select_cor_table**. Before returning, **select_cor_table** itself internally waits for the end of the jump.
 - If **load_correction_file** is called after **load_program_file**, then:
 - **load_correction_file** automatically executes a **select_cor_table(HeadA, HeadB)** with the last used values for **HeadA** and **HeadB**
 - **load_correction_file** positions the galvanometer scanners with an internal jump at the currently set jump speed to the output position specified there
 - If **load_correction_file** is called prior to **load_program_file**, then the correction table is only saved. There can be no internal jump to the intended output position.

(1) Correction files can also be loaded before the **RTC6 files** have been loaded by **load_program_file**.



1to1 Correction Tables

1to1 correction files

- Are not assigned to a particular scan system
- In general, serve test purposes only
- Alternatively, can be loaded by **load_correction_file** with parameter **Name = NULL**.
 - According to the further **Dim** parameter a 2D or 3D 1to1 correction table is being generated internally.

Because some user programs require a file name and do not accept **NULL** as the name, the 1to1 correction file **Cor_1to1.ct5** is supplied within the RTC6 Software Package. This contains a calibration factor of value 0. If a user program strictly needs an “authentic” calibration factor, a corresponding value can be specified with **CorrectionFileConverter.exe**, see **Section “Folder CorrectionFileConverter”, page 31** (button **Show File Header** > field ‘Field Calibration [Bit/mm]’).

Inverse Tables

The ct5-correction file format supports “inverse tables”. These are required for back transforming actual scan head positions to the associated **Image field** coordinates.

For further information, see:

- Parameter **11**, see [page 178](#)
- [Chapter 8.1.3 “Monitoring the Positioning”, page 213](#)
- [Section “Converting Correction Files”, page 179](#)

ct5 Correction File Header

The ct5-correction file header contains 16 parameters, see following table.

These can be read out and thus directly incorporated into a user program:

- from the currently loaded correction tables by [get_table_para](#)
- from assigned correction tables by [get_head_para](#)

Notes

- With ctb-correction files, some parameter values can exclusively be taken from its associated [*_ReadMe.txt](#) file. These must be transferred manually to the user program.
- A ct5 file created by converting another ctb file, see [Section "Converting Correction Files", page 179](#), does not contain all parameter data.
- A 1to1 correction file does not contain all parameter data.
- Parameters 3...7 are only relevant for 3D marking. In 2D correction tables, they are 0.
- For the same 3-axis scan system, the ABC coefficients (parameter 5...7) in ctb-correction files and ct5-correction files are identical.

Parameter ^(a)	Description
0	Type of the correction table. <ul style="list-style-type: none"> • = 0.0: 2D correction table • = 1.0: 3D correction table
1	Calibration factor K_{xy} [bit/mm]. See Chapter 7.3.2 "Image Field Size and Image Field Calibration", page 166 .
2	Focal length or working distance [mm]. <ul style="list-style-type: none"> • For a configuration with a scan objective: the effective focal length of the objective [mm]. • For a configuration without a scan objective: the working distance A [mm]. A = 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. Compensates the pyramid-shaped Image field change which exists in the z direction of 3D markings.
4	Stretch factor for the y direction. See parameter 3.
5	Coefficient A of the polynomial for z axis control, see page 169 : offset part, ± 26 Bit.
6	Coefficient B of the polynomial for z axis control, see page 169 : linear part, ± 11 Bit.
7	Coefficient C of the polynomial for z axis control, see page 169 : square part, ± 4 Bit.
8	Number of the correction file. With correction files supplied by SCANLAB, the parameter corresponds to the number in the file name (for example, 145 for <code>D2_145.ct5</code> or <code>D3_145.ct5</code>).

Parameter ^(a) (cont'd.)	Description (cont'd.)
9	<p>Differentiation between correction with or without an F-Theta objective. The following applies: Parameter = $10 \times P_{Obj} + P_{Typ}$ with</p> <ul style="list-style-type: none"> • $P_{Obj} = 0$: Correction without F-Theta objective • $P_{Obj} = 1$: Correction with F-Theta objective • If correction with F-Theta objective: <ul style="list-style-type: none"> $P_{Typ} = 0.0$: without distortion data $P_{Typ} = 1.0$: with F-Theta's F-stop progression condition $P_{Typ} = 2.0$: with image height table
10	<p>Indicator for the source of the correction file. The following applies: Parameter = $1000 \times P_{Orig} + P_{Ver}$ with</p> <ul style="list-style-type: none"> • $P_{Orig} = 10000$: originally calculated file • $P_{Orig} = 20000$: converted from <code>ctb</code> file • $P_{Orig} = 30000$: reconstructed from <code>txt</code> file By manipulating a correction file using correction programs available from SCANLAB, P_{Orig} is increased by 1 in each case. – P_{Ver} = Version number of the program used to create the correction file
11	<p>Information about the inverse table. The following applies: Parameter = $P_{Exist} + 2 \times P_{Calc}$ with</p> <ul style="list-style-type: none"> • $P_{Exist} = 1.0$: valid inverse table is present • $P_{Exist} = 0.0$: no valid inverse table present • If valid inverse table is present: <ul style="list-style-type: none"> $P_{Calc} = 0$: inverse table calculated ab initio $P_{Calc} = 1$: inverse table numerically calculated
12	<p>Angle calibration of the scan system. Mechanical angle deflection in [\pm °] at 96% of the maximum control.</p>
13	<p>Code for the scan head geometry used for the calculation (for internal use only), for example,</p> <ul style="list-style-type: none"> • = -1.0: unknown geometry (for example, for a table converted from a <code>ctb</code> file) • = 0.0: standard geometry
14	<p>Indicator for whether an additional protective window has been taken into account. The following applies: Parameter = $1,000,000 \times P_T + 1,000 \times P_I$ with</p> <ul style="list-style-type: none"> • P_T = Protective window thickness in mm (max. 2 decimal places) • P_I = Refraction index (max. 3 decimal places) <p>Example: The value 3,521,450.0 corresponds to a protective window thickness of 3.52 mm and a refraction index of 1.450.</p>
15	<p>Indicator for whether the Image field size has been limited in the correction file.</p> <ul style="list-style-type: none"> • = 0.0: without field size limit • = 2.0: with field size limit

(a) Numbering according `get_table_para` and `get_head_para`.



Converting Correction Files

The RTC6 Software Package includes the program [CorrectionFileConverter.exe](#) for converting **ctb** correction files to **ct5** correction files and vice versa. The corresponding manual is supplied in English and German.

When converting a **ctb**-correction file into a **ct5**-correction file, the **ct5**-correction file header receives a corresponding origin indicator (parameter [10, page 178](#), $P_{\text{Orig}} = 20000$).

Such conversions do *not* produce fully complete **ct5**-correction files because the file header lacks several parameters. These can be subsequently entered manually by [CorrectionFileConverter.exe](#) (via button **Show File Header**).

Moreover, it may happen that the inverse correction table to be calculated numerically during the conversion does not cover the entire **Image field** or cannot be calculated. In this case, a 1to1 correction table is inserted instead, see also parameter [11, page 178](#).

In contrast, converting a **ct5**-correction file into a **ctb**-correction file results in a fully complete **ctb**-correction file.

Parameter information from the **ct5**-correction file header are:

- no longer contained in a **ctb**-correction file for 2D correction table
- contained partially in a **ctb**-correction file for 3D correction tables

7.3.6 Output Values to the Scan System

Calculation

After **Microstepping** (1), the following corrections are applied in the order given to calculate the output values from the current x, y, and z coordinate values.

- (1) The split-up of vectors and arcs to **Microsteps**.
- (2) A wobble motion that has been set, see [Chapter 8.4 "Wobble Mode", page 231](#), for example, by **set_wobble_mode**.
- (3) A global coordinate transformation in the virtual **Image field**, for example, by **set_matrix**, **set_angle** and **set_offset** with **HeadNo** = 4, see [Section "Coordinate Transformations in the Virtual Image Field", page 168](#)
- (4) A Processing-on-the-fly correction that has been set, see [Chapter 8.6 "Processing-on-the-fly", page 241](#), for example, by **set_fly_x** or **set_fly_x_pos**.
- (5) A 2D coordinate transformation for aligning the scan system relative to the **Image field**, see [Chapter 8.2 "Coordinate Transformations", page 223](#), for example, by **set_matrix**, **set_scale**, **set_angle** and corresponding list commands.
- (6) A 3D coordinate shift that has been set, see [Chapter 8.5.2 "3D Scan Systems", page 236](#), for example, by **set_offset_xyz** or **set_offset_xyz_list**.
- (7) Coordinate values that exceed the real **Image field** range (virtual **Image field**), see [Chapter 7.3.3 "Virtual Image Field", page 168](#), are clipped to the boundary values of the real 20-bit **Image field**.
- (8) A 2D/3D **image field** correction
 - correspondingly to the correction table that has been assigned by **select_cor_table** or **select_cor_table_list**, see [Chapter 7.3.5 "Image field Correction and Correction Tables", page 172](#)
 - and a focus shift that has been set for example, by **set_defocus** or **set_defocus_list**.

- (9) A gain correction and offset correction for the x galvanometer scanner and y galvanometer scanner. These can be set:
 - explicitly by **set_hi**, see [Section "Customer-Specific Calibration", page 277](#)
 - automatically by **auto_cal**, see [Chapter 8.10 "Automatic Self-Calibration", page 274](#)

Notes

- Overflowing values are always clipped to the boundary values of the maximum possible value range.
- A complete 3D calculation is always performed. If the **Option "3D"** is not enabled, z values are just not outputted.

Value Ranges

For all scan system axes, signed 20-bit values ($-524,288 \dots +524,287$) are outputted. This also applies to values which the scan system returns to the RTC6 PCIe Board, see also:

- [Section "RTC4 Compatibility Mode", page 167](#)
- [Section "RTC5 Compatibility Mode", page 167](#)
- [Chapter 4.5.2 "XY2-100 Converter \(Accessory\)", page 68](#)

Precalculation and Diagnosis

With **get_galvo_controls** the output values resulting for the given coordinates and setting parameters in the current configuration (correction table, coordinate transformations) can be determined without the galvanometer scanners moving.



Clock Overruns

The 10 μ s clock cycle might not always suffice for calculating all data required by the computation-intensive **Jump Commands**, **Mark Commands** and **Arc Commands** if several of the available command options are utilized simultaneously – for example, simultaneous control of two scan systems, wobble motion, coordinate transformation in the virtual **Image field**, Processing-on-the-fly for two axes (correction by **McBSP interface**), "Automatic Laser Control", para vectors, data recording, "Variable **Polygon Delay**", short list commands (for example, in a **Polyline**), 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 μ s clock cycle, possibly including output to the scan head. The galvanometer scanner motion might also could pause for 10 μ s.

7.3.7 Status Monitoring and Diagnostics

For status monitoring and diagnostic purposes, `get_value` or `get_values` can be used to read out a variety of signals:

- A 20-bit status word which is returned by the scan system, see [Section "Status Information Returned from the Scan System", page 182](#)
- The current LASERON signal
- The current Cartesian control values (that is, the so-called "sample values" common to both scan head connectors, see [2](#), [3](#) and [4](#)).
- The control values specific to each scan head connector which take into account
 - any coordinate transformations defined by `set_matrix`, `set_scale`, `set_angle` or by the corresponding list commands, see [5](#).
 - any z coordinate offset defined by `set_offset_xyz` or `set_offset_xyz_list`, see [6](#).
 - any focal length offset defined by `set_defocus` or `set_defocus_list` and any loaded correction table, see [8](#) and [9](#).

Each of these signals can be recorded on the RTC6 PCIe Board for a longer time period by `set_trigger/set_trigger4` – with a selectable sample period. After that, `get_waveform` can be used to transfer them to the PC for analysis.

By `set_trigger(Period Bit #31 = 1)` (ring buffer functionality), practically arbitrary amounts of data can be recorded. The recorded values can be transferred to the PC in blocks during list execution using `get_waveform_offset`. After the measurement data memory has reached its capacity, measurement value recording automatically starts from the beginning.

The current status and progress of a measurement session started with `set_trigger/set_trigger4` can be queried by `measurement_status`.

The values returned by the scan system are always 20-bit values.

Status Information Returned from the Scan System

The scan system transmits the following signals to the RTC6 PCIe Board every $10 \mu\text{s}$ via the SL2-100 protocol:

- A 20-bit status word. It can mean the following data types:
 - The actual 20-bit status word (the upper 16 bits are identical to the 16-bit status word of the XY2-100 protocol ([XY2-100 status word](#)), for a description see `get_head_status`).
 - Only for iDRIVE scan systems⁽¹⁾, see [Chapter 8.1 "iDRIVE Functions", page 211](#): a different data type selectable with `control_command`.

The 20-bit status word can:

- be read out by `get_value/get_values`
- be recorded by `set_trigger/set_trigger4`

- 6 additional status bits.

With iDRIVE scan systems⁽¹⁾ with SL2-100 protocol, the status bits (PowerOK, TempOK and PosAck; per axis) are transferred in parallel to and independently from the 20-bit status word. Therefore, these status bits can even be used to monitor the scan system (see [Section "Automatic Suppression of Laser Control Signals", page 186](#)) if, for example, an "Automatic Laser Control" (see [Chapter 7.4.9](#) "Automatic Laser Control", page 196) is active that requires a different return data type.

The 6 additional status bits can be read out by `get_head_status`.

Notes

- Scan systems with XY2-100 protocol require the use of the [XY2-100 Converter \(Accessory\)](#).
- For iDRIVE scan systems⁽¹⁾ with XY2-100 protocol, the actual [XY2-100 status word](#) must be set as the to-be-returned data type (default type after switching on).

(1) See Glossary entry on [page 26](#).

7.4 Laser Control

At certain output ports, the RTC6 PCIe Board outputs signals which can be used for controlling various laser types ("laser control signals" LASERON, LASER1, LASER2). These output ports are at the:

- **LASER Connector**, see [Chapter 4.6.1 "LASER Connector", page 72](#)
- **EXTENSION 2 socket connector**, see [Chapter 4.6.3 "EXTENSION 2 Socket Connector", page 79](#)

The laser control mode is set by `set_laser_mode`:

- The **CO₂ Mode** (laser mode 0; default after `load_program_file`) is designed for controlling a CO₂ laser.
- The **YAG modes** (laser mode 1, 2, 3, 5) are designed for controlling lasers from the Nd:YAG family (and related).
- **Laser Mode 4** and **Laser Mode 6** are designed for controlling free-running lasers.⁽¹⁾

All laser control signals are TTL signals. By `set_laser_control`, it is set whether they are active-HIGH or active-LOW, see also [Chapter 4.6.1 "LASER Connector", page 72](#). Their current setting can be queried by `get_startstop_info` (Bit #13).

For the maximum current load values, see [Chapter 15 "Technical Specifications – RTC6 PCIe Board", page 854](#).

7.4.1 Enabling, Activating and Switching Laser Control Signals

All laser control signals are suppressed:

- After a hardware reset and
- After initialization with `load_program_file`.

Then, all laser control signal output ports (LASERON, LASER1 and LASER2) are in high impedance mode.

Before laser control signals can be outputted at all, their polarity must first be set by `set_laser_control`. This cancels the tristate state at the same time (= "global unblock"). By default, LASERON and LASER1/LASER2 are set to their respective "Off" levels.

The tristate state is only set again:

- after a hardware reset (restart)
- a call of `load_program_file`

Furthermore, real laser control signals must be enabled:

- either simultaneously by `set_laser_control(Bit #2 = 0)` or
- afterwards with the separate command `enable_laser`

These can be suppressed again by `disable_laser` or `pause_list`. In both cases, all laser control signals are set to their respective "Off" levels.

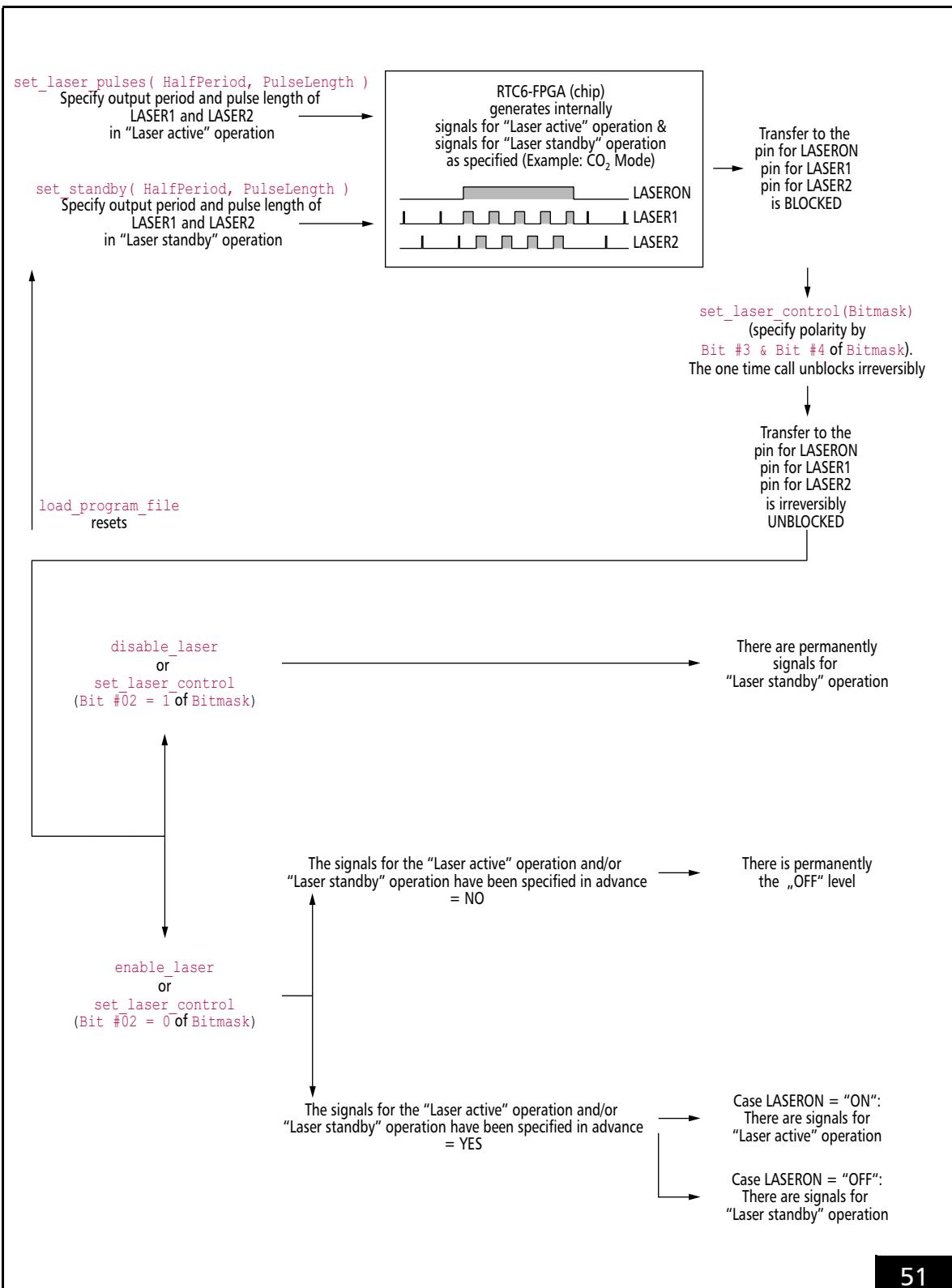
By `get_startstop_info` (Bit #9, Bit #10, Bit #13, Bit #14) it can be read out:

- The "global unblock" by `set_laser_control`
- The polarities of LASERON and LASER1/LASER2
- The enabling of the [Signals for "Laser Active" Operation](#) by `enable_laser`

General notes on the laser control signals can be found in [Section "Signals for "Laser Active" Operation", page 185](#) and [Section "Signals for "Laser Standby" Operation", page 185](#). Mode-specific details depend on the set laser mode, see [Chapter 7.4.3 "CO₂ Mode", page 187](#) to [Chapter 7.4.8 "Pulse Picking Laser Mode", page 195](#).

The assignment of the laser control signals to the respective output pins at the **LASER Connector** can be configured, see [Chapter 7.4.2 "Configuring the LASER Connector", page 186](#).

(1) For synchronization to externally controlled free-running lasers, see [Chapter 7.4.10 "Synchronization of the RTC6 Clock Cycle and an External Clock Signal", page 206](#).



RTC6 board hardware and laser control-related RTC6 commands.

Signals for "Laser Active" Operation

By `set_laser_pulses`, `set_laser_pulses_ctrl` or `set_laser_timing`, the Signals for "Laser Active" Operation can be

- activated: `HalfPeriod` $\neq 0$ and `PulseLength` $\neq 0$
- deactivated: `HalfPeriod` = 0 and/or `PulseLength` = 0

Even if the Signals for "Laser Active" Operation have been enabled and activated, they are *only outputted at the output ports, if they are switched on by further commands*.

They are automatically switched on, when:

- **Mark commands** are called, see [Chapter 7.1.1 "Marking with Vector Commands and "Arc" Commands", page 135](#)

They are automatically switched off, when:

- a **Mark command** is followed by a normal non-mark command
- a list is terminated by `set_end_of_list` or `stop_execution`
- a list is temporarily suspended by `set_wait`, `pause_list` or `stop_list`

In the latter case, the Signals for "Laser Active" Operation are switched on again if the list is continued by `release_wait` or `restart_list`.

They can also be switched on and off within a list with `laser_signal_on_list` and `laser_signal_off_list` or outside a list with `laser_signal_on` and `laser_signal_off` for an unlimited time.

Pulse Completion

Whether a modulation pulse (Q-Switch pulse) started with the LASERON signal switched on is still completely executed or cut off at LASER1 if it has not yet been fully processed when the LASERON signal is switched off, can be set by `set_laser_control(Bit #0)`.

Signals for "Laser Standby" Operation

By `set_standby` or `set_standby_list`, the Signals for "Laser Standby" Operation can be

- activated: `HalfPeriod` $\neq 0$ and `PulseLength` $\neq 0$
- deactivated: `HalfPeriod` = 0 and/or `PulseLength` = 0

The Signals for "Laser Standby" Operation are continuously outputted at the output ports after their activation (without further commands), if no signals for the "Laser active" operation are switched on.

If the Signals for "Laser Active" Operation are deactivated (for example, by `PulseLength` = 0), there is no changeover. Then the Signals for "Laser Standby" Operation are continuously outputted even during the execution of **Mark commands**.

If the Signals for "Laser Standby" Operation are deactivated, all output ports are set to the "Off" level in the "Laser standby" mode.

If activated signals for the "Laser standby" operation are to be deactivated when stopping a list with `pause_list` (here, only the Signals for "Laser Active" Operation are automatically deactivated), users must explicitly initiate this by calling `set_standby(PulseLength = 0)`.

The current "Laser standby" parameters that may have been changed within a list can be read out by the control command `get_standby`.

Pulse Completion

With the Signals for "Laser Standby" Operation, pulse completion, see [Pulse Completion, page 185](#), is not supported.

Automatic Suppression of Laser Control Signals

Case: Scan System Status Errors

By `set_laser_control` (Bit #16...Bit #27), it can set that the laser control signals are to be automatically suppressed when the corresponding scan-system status signal (PowerOK, TempOK, PosAck of axis X/Y of head A/B) indicates an error (that is, is 0; "NOK").

As soon as at least one of the specified status signals is 0, then:

- Output of the laser control signals are automatically interrupted. They 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`)
- If accordingly set by `set_laser_control(Bit #28 = 1)`, a `stop_execution` is automatically executed (the list stops, laser control signals get permanently switched off)
- If accordingly set by `set_laser_control(Bit #29 = 1)` in addition, the `stop_execution` is forwarded as /Master-STOP (see Figure 68) to all Master/Slave connected RTC6 boards. Whether the signal is effective there can be set individually for each RTC6 board with `master_slave_config`. This ensures that the laser is automatically switched off even if the error occurs on an RTC6 board that is not intended for controlling the laser.

Case: Galvanometer Scanner Position Exceedances

`range_checking` can be used to define that the laser control signals are to be automatically suppressed as soon as a galvanometer scanner exceeds a predefined range limit.

They are automatically switched on again as soon as the next `Mark command` starts within the permitted range. This means that an interrupted `Polyline` remains suppressed for the rest of the `Polyline`.

7.4.2 Configuring the LASER Connector

`LASER Connector` pin (01), (02) and (09) can be configured by `config_laser_signals` and `config_laser_signals_list`, see Figure 17.

If you employ a variety of lasers or laser operational modes, then these commands might eliminate the need to configure at the hardware level (that is, using various cables and switches).

Whereas the default setting (for normal markings) 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 Output Mode` with pulse picking.

For other operational modes, you could also 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).

The following description of the various laser modes applies to the default setting for laser control signal output.

7.4.3 CO₂ Mode

`set_laser_mode(0)` sets the CO₂ Mode ("Laser Mode 0").

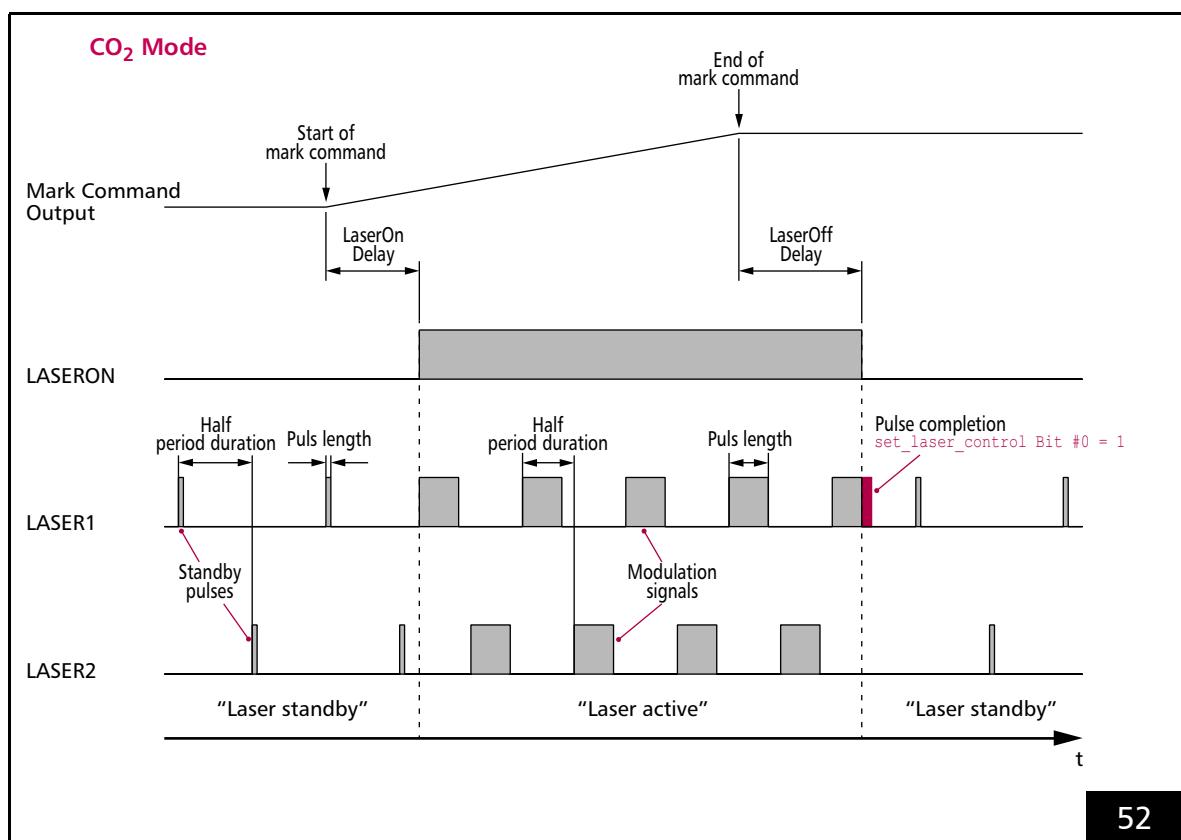
The timing diagram, see Figure 52, shows the corresponding signals using the example of an isolated mark command.

For "laser active" operation:

- The LASERON signal is switched on
- 2 alternating modulation signals are outputted at the LASER1 and LASER2 output port. Their pulse length and period duration can be defined by `set_laser_pulses`, `set_laser_pulses_ctrl` or `set_laser_timing`.

For "laser standby" operation:

- The LASERON signal is switched off
- Alternating standby pulses are outputted at the LASER1 and LASER2 output port. Their pulse length and period duration can be defined by `set_standby` or `set_standby_list`.



Timing diagram of the signals in CO₂ Mode (with active-HIGH laser control signals).
Example: isolated mark command.



Notes

- LASER1 signals and LASER2 signals are
 - activated by: `HalfPeriod ≠ 0` and
`PulseLength ≠ 0`
 - deactivated by: `HalfPeriod = 0` and/or
`PulseLength = 0` (default setting after
`load_program_file`)
- The LASER2 signal is phase-shifted by half a signal period in relation to the LASER1 signal. It can be used for the control of a second laser tube. By `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 periods.
- For the LASER1 signals and LASER2 signals, *half* of the output period must be specified.
- `set_laser_pulses` and `set_laser_timing` are delayed short list commands. They can also be used to change the laser parameters between two `Mark commands`.
The time base for the signals is always:
 - 1/64 μ s in `RTC6 Standard Mode`,
see also `Section "RTC6 Standard Mode"`,
`page 167`
 - 1/8 μ s in `RTC4 Compatibility Mode`,
see also `Section "RTC4 Compatibility Mode"`,
`page 167`
- `Pulse Completion`, `page 185` affects also LASER2 signals.

7.4.4 YAG Modes 1, 2, 3, 5

`set_laser_mode([1, 2, 3 or 5])` sets one of the YAG modes.

The timing diagram, see [Figure 53](#), shows the corresponding signals using the example of an isolated mark command.

In each YAG mode, for “laser active” operation:

- The LASERON signal is switched on
- A *Q-Switch signal* is outputted at the LASER1 output port, see [Q-Switch Signal, page 189](#).
- A adjustable *FirstPulseKiller signal* is outputted at the LASER2 output port, see [FirstPulseKiller Signal, page 189](#).

In each YAG mode, for “laser standby” operation:

- The LASERON signal is switched off
- Standby pulses are outputted at the LASER1 output port. Pulse length and period duration of the standby pulses can be set by `set_standby` or `set_standby_list`.

Notes

- LASER1 signals are
 - activated by: `HalfPeriod ≠ 0` and `PulseLength ≠ 0`
 - deactivated by: `HalfPeriod = 0` and/or `PulseLength = 0` (default setting after [load_program_file](#))

Q-Switch Signal

The Q-Switch signal serves to control the quality switch of the laser. The Q-Switch period duration and pulse length are set by `set_laser_pulses`, `set_laser_pulses_ctrl` or `set_laser_timing`.

Notes

- See also [Section “Pulse Completion”, page 185 \(Signals for “Laser Active” Operation\)](#).
- See also [Section “Pulse Completion”, page 185 of \(Signals for “Laser Standby” Operation\)](#).

FirstPulseKiller Signal

The FirstPulseKiller signal serves to signal the first laser pulses of a pulse sequence, if they do not correspond to the usual continuous power.

This signal is only provided. The laser itself must respond appropriately. The FirstPulseKiller signal is outputted automatically together with the LASERON signal.

The length of the FirstPulseKiller signal is set with `set_firstpulse_killer` or `set_firstpulse_killer_list`.

Differences Between the YAG Modes

YAG Mode 1, YAG Mode 2, YAG Mode 3 and YAG Mode 5 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 53](#).

After the Q-Switch delay has expired, the first Q-Switch pulse starts. The Q-Switch delay value can be specified by `set_qswitch_delay` or `set_qswitch_delay_list`; alternatively by selecting the YAG mode:

- YAG Mode 1: 0
- YAG Mode 2: length of the FirstPulseKiller signal
- YAG Mode 3: 10 μ s
- YAG Mode 5: value from `set_qswitch_delay` or `set_qswitch_delay_list`

By `set_laser_mode`, the Q-Switch delay is merely *preset*. Therefore, in YAG Mode 1, YAG Mode 2, YAG Mode 3, too, the Q-Switch delay can be subsequently changed by `set_qswitch_delay` or `set_qswitch_delay_list`.

In YAG Mode 2, the Q-Switch delay is also adjusted accordingly with each `set_firstpulse_killer` or `set_firstpulse_killer_list`.



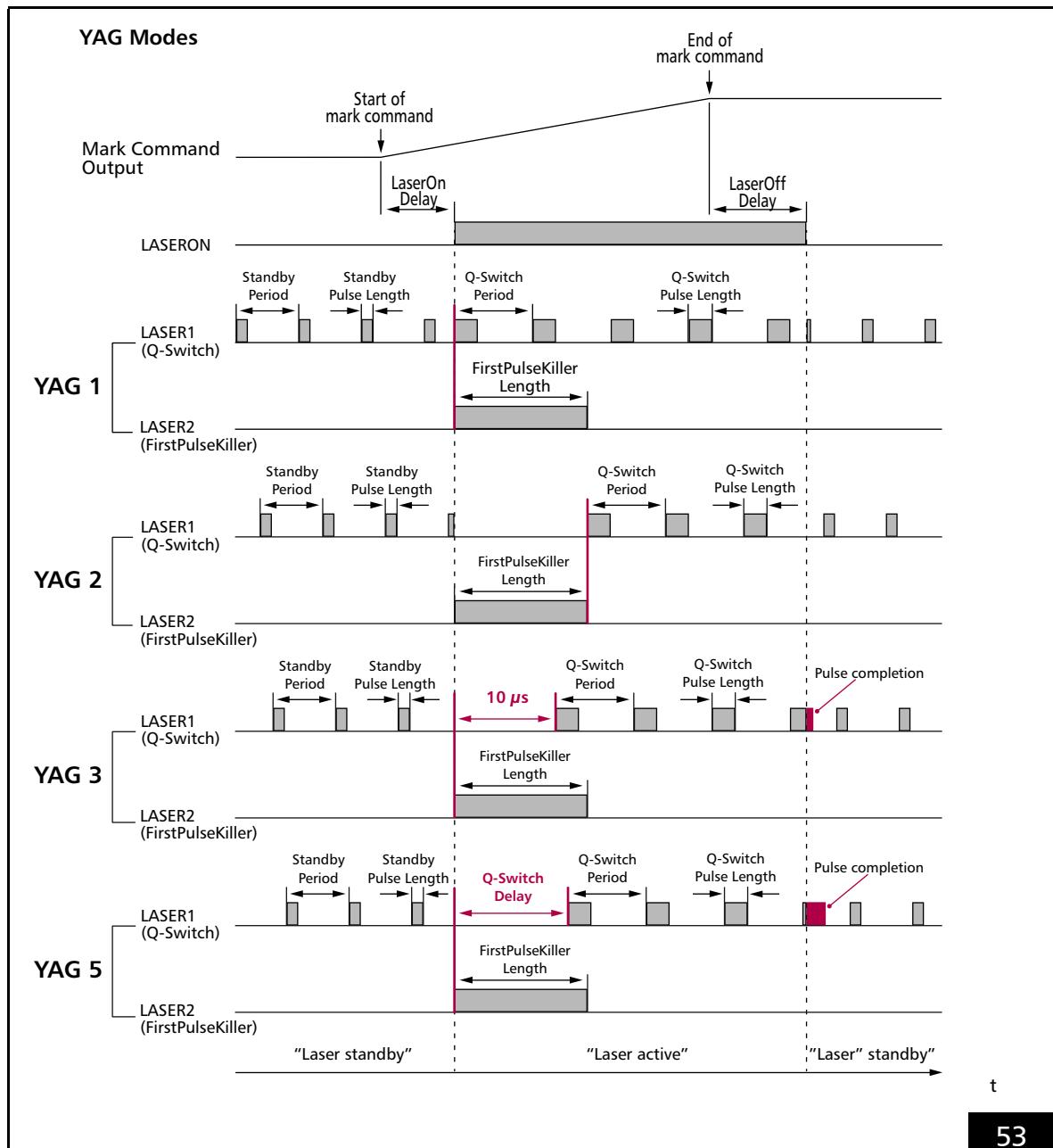
Lamp Current (Laser Power)

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 at the **LASER Connector**, see [Figure 17](#). **ANALOG OUT2** is also available at the **MARKING ON THE FLY** socket connector, see [Figure 25](#).

To define the analog output signal, the control command **write_da_x** and the delayed short list command **write_da_x_list** are provided.

Alternatively, the lamp current can be controlled digitally via the 8-bit digital output port (at the **EXTENSION 2** socket connector, see [Figure 24](#)), see also [Section "8-Bit Digital Output Port", page 80](#). The commands for setting the 8-bit output port are **write_8bit_port**, **write_8bit_port_list** or **write_port_list**.

When the lamp current is changed, list execution can be halted by **long_delay** until a constant laser power has been achieved.



Timing diagram of the signals in the YAG modes (with active-HIGH laser control signals). Standby signals are not synchronized with the "laser-active" signals and can have arbitrary phase alignments.
Example: isolated mark command.

7.4.5 Laser Mode 4

`set_laser_mode(4)` sets the **Laser Mode 4**.

The timing diagram, see [Figure 54](#), shows the corresponding signals using the example of an isolated mark command.

At LASER1 output port, for "laser active" operation as well as for "laser standby" operation standby pulses are outputted continuously. Pulse length and period duration of the standby pulses can be set by `set_standby` or `set_standby_list`.

For "laser active" operation:

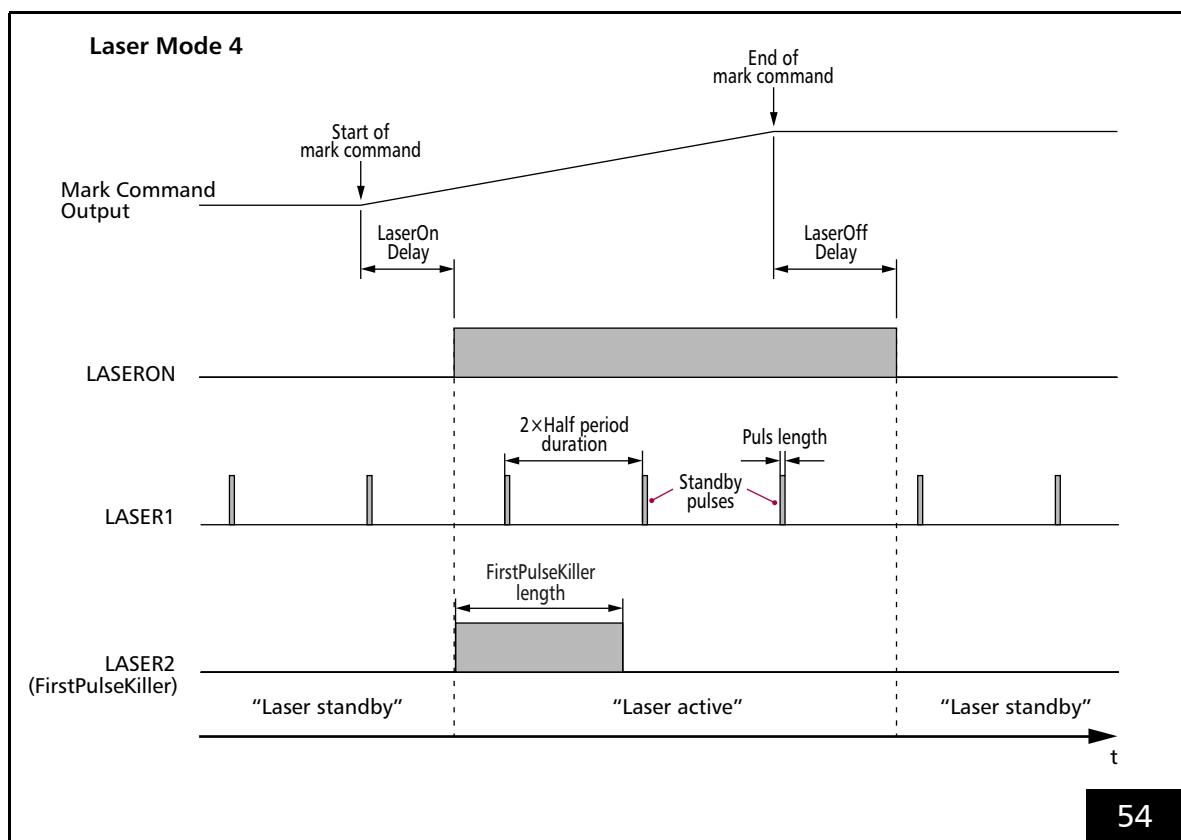
- The LASERON signal is switched on
- A programmable FirstPulseKiller signal is outputted at the LASER2 output

For "laser standby" operation:

- the LASERON signal is switched off
- and the LASER2 signal is switched off

Notes

- LASER1 signals are
 - activated by: `HalfPeriod ≠ 0` and `PulseLength ≠ 0`
 - deactivated by: `HalfPeriod = 0` and/or `PulseLength = 0` (default setting after `load_program_file`)
- The FirstPulseKiller signal is started together with the LASERON signal automatically. The length of the FirstPulseKiller signal is set by `set_firstpulse_killer` or `set_firstpulse_killer_list`.
- **Laser Mode 4** is used for some fiber lasers.



54

Timing diagram of the signals in **Laser Mode 4** (with active-HIGH laser control signals).
Example: isolated mark command.

7.4.6 Laser Mode 6

`set_laser_mode(6)` sets the **Laser Mode 6**.

Laser Mode 6 is like **Laser Mode 4** and is provided for those free-running lasers whose gate signals (LASERON) must *not* be changed during the duration of a pulse.

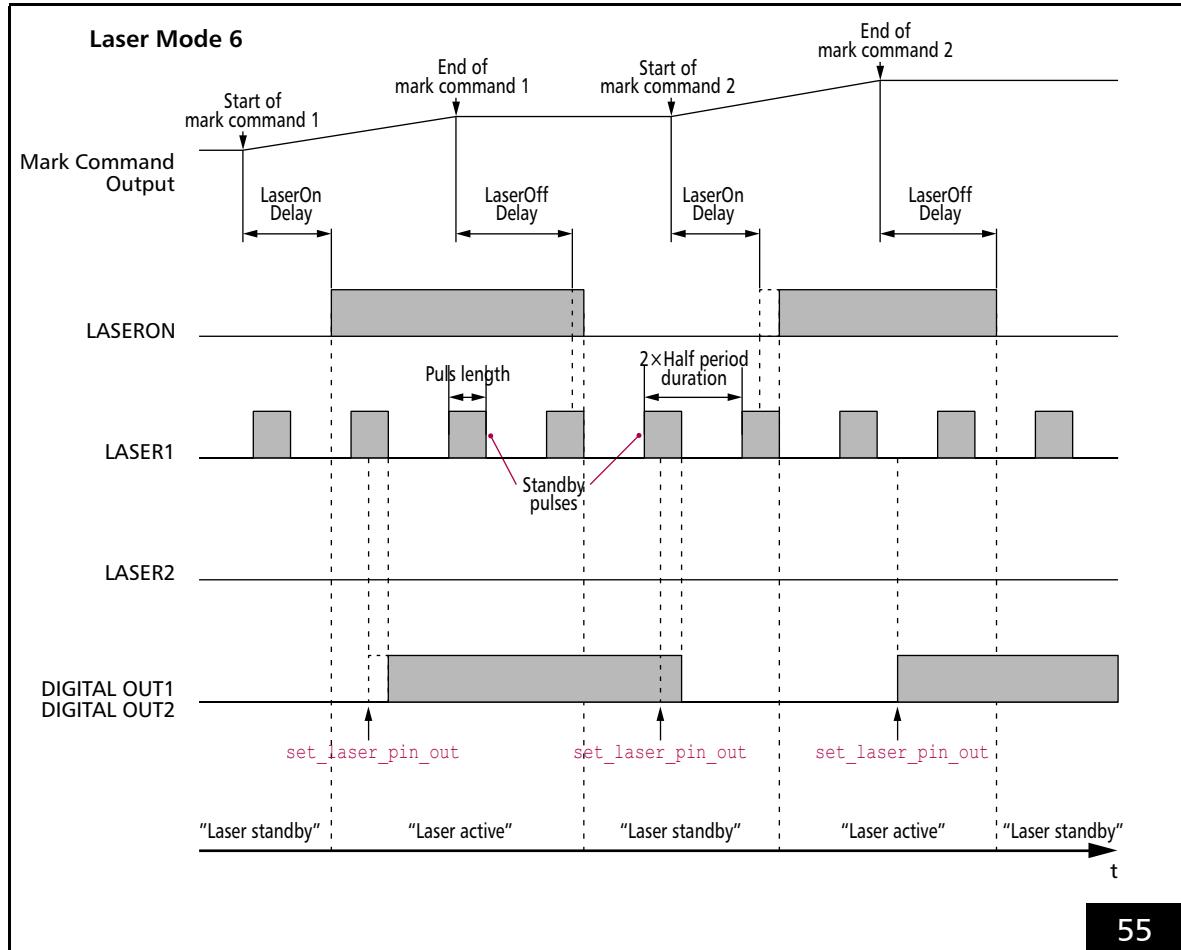
The timing diagram, see [Figure 55](#), shows the corresponding signals using the example of an isolated mark command.

Laser Mode 6 signals and **Laser Mode 4** signals are the same (see also Notes there, [page 192](#)) with the following exception:

- As long as a standby pulse is active, switching of the LASERON signal is delayed accordingly. LASERON switches 5/64 μ s after the end of the standby pulse.

Notes

- No LASER2 signal is outputted.
- The delay of the switching time when a standby pulse is still active is also valid for switching the output values at the 2-bit digital output port (DIGITAL OUT1 and DIGITAL OUT2) by `set_laser_pin_out`, `set_laser_pin_out_list` or `write_port_list`, see [Figure 55](#).



Timing diagram of the signals in **Laser Mode 6** (with active-HIGH laser control signals). Example: isolated mark commands.



7.4.7 Softstart Mode (not yet implemented)

Not implemented yet.

7.4.8 Pulse Picking Laser Mode

By `set_pulse_picking` or `set_pulse_picking_list`, the **Pulse Picking Laser Mode** can be set. The laser control timing diagram in Figure 56 shows the corresponding signals.

For "laser active" operation:

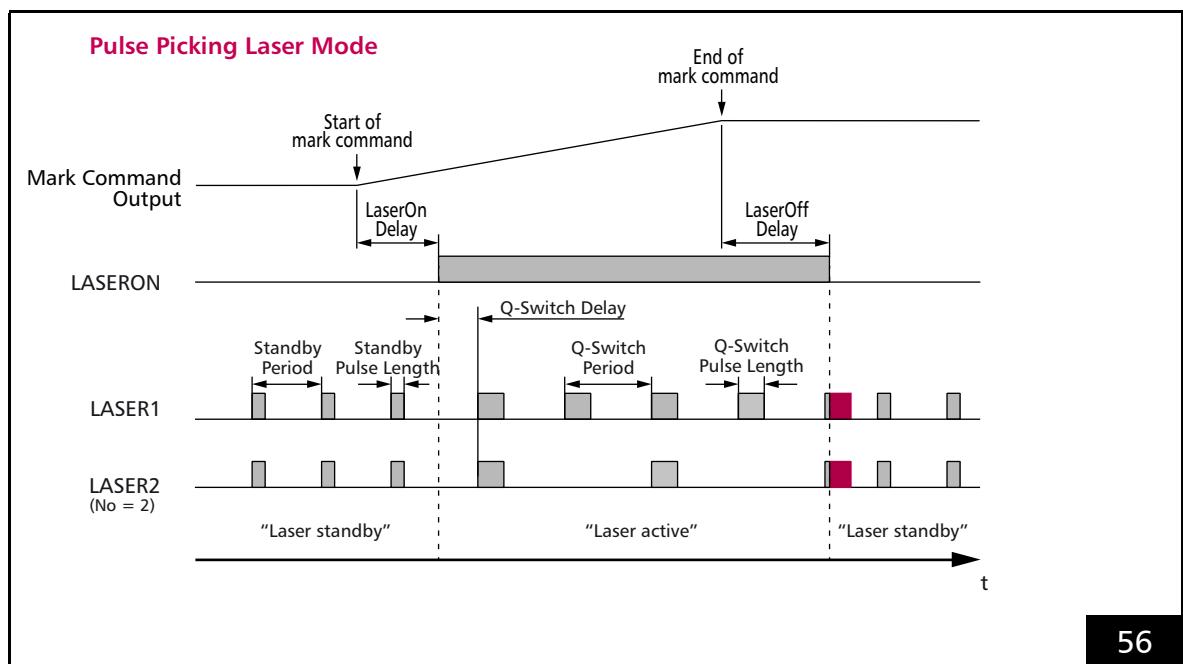
- The LASERON signal is switched on
- A modulation signal is provided at the LASER1 output with pulse length and period duration that can be defined by `set_laser_pulses`, `set_laser_pulses_ctrl` or `set_laser_timing`
- At the LASER2 output every No^{th} pulse of the signals is outputted at the LASER1 output (in phase). No is set with `set_pulse_picking` or `set_pulse_picking_list`

For "laser standby" operation:

- 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`. Standby pulses cannot be "pulse-picked".

Notes

- With `set_laser_control` (Bit #7 = 1) the Pulse Picking signal at LASER2 can be set to a constant length independent of the LASER1 signal, which is set with `set_pulse_picking_length`.
- `set_pulse_picking` and `set_pulse_picking_list` overwrite a laser mode previously set with `set_laser_mode`. Vice versa, `set_laser_mode` switches off the **Pulse Picking Laser Mode**.
- For the LASER1 signals, half the period duration must be specified.
- A Q-Switch delay is effective, see also [Section "Differences Between the YAG Modes", page 189](#).
- A FirstPulseKiller signal is not outputted.
- The setting sequence of the LASER1 signals and the **Pulse Picking Laser Mode** is irrelevant.



Timing diagram of the signals in **Pulse Picking Laser Mode** (with active-HIGH laser control signals and $No = 2$).
Example: isolated mark command.

7.4.9 "Automatic Laser Control"

By `set_auto_laser_control`, automatic readjustment of the **Signals for "Laser Active" Operation** – even dynamically during execution of **Mark commands** can be achieved.

The `Ctrl` parameter determines the output port and the `Value` parameter the output value for 100% power at the target (set) mark speed at the **Image field** center (for the other parameters, see command description).

This can be used to compensate for variations in laser energy input caused by a changing power density, spot size or processing speed.

For "Automatic Laser Control", a position-, speed- or vector-controlled correction of the laser control signals can be activated and combined.

- *Position-dependent* laser control, see **Section "Position-Dependent Laser Control", page 198**, allows compensation of radial laser energy input variations during execution of **Mark commands**. Such variations may be caused, for example, by a objective edge diminution or different incidence angles of the laser beam.
- *Speed-dependent* laser control, see **Section "Speed-Dependent Laser Control", page 200**, allows compensation of laser energy input variations during execution of **Mark commands** that resulting from an uneven galvanometer scanner movement (acceleration phase and deceleration phase, time-dependent **Mark commands**). In addition, the galvanometer scanner speed can also be transformed back into a mark speed depending on the position.
- *Encoder speed-dependent* laser control, see **Section "Encoder-Speed-Dependent Laser Control", page 203**, allows the laser energy input to be controlled based on the currently present encoder speed.

- Speed-dependent laser control and encoder speed-dependent laser control can also be combined with each another, if galvanometer scanner and workpiece move simultaneously.

- *Vector-defined* laser control, see **Section "Vector-Defined Laser Control", page 205**, allows linear readjustment of a signal parameter along parameterized mark vectors or jump vectors (see **Section "[*]Para[*] Commands", page 138**).

This signal parameter can be combined with the laser power control if the control parameter `Ctrl` matches `Ctrl` from `set_auto_laser_control` (the current parameter `Para` dynamically replaces the 100% value from `set_auto_laser_control`) or as an independent control (for example, as defocus).

Selectively, "Automatic Laser Control" adjusts one of the following signal parameters:

- 12-bit output value at the **ANALOG OUT1** or **ANALOG OUT2** output port of the **LASER Connector**, see also **Section "12-Bit Analog Output Port 1 and 2", page 73**
- Output value at the 16-bit digital output port of the **EXTENSION 1** socket connector, see also **Section "16-Bit Digital Input Port and 16-Bit Digital Output Port", page 77**
- Output value at the 8-bit digital output port of the **EXTENSION 2** socket connector, see also **Section "8-Bit Digital Output Port", page 80**
- Pulse length (`PulseLength`) or output period (`HalfPeriod`) of the laser control signals **LASER1** and **LASER2**

The automatically readjusted value can be recorded by `set_trigger/set_trigger4` (Signal `n = 24`).

Notes

- “Automatic Laser Control” does *not* compensate for an explicit change in mark speed between two **Mark commands** caused by a **set_mark_speed** call.
- “Automatic Laser Control” and **Pixel mode** should not affect the same output port at the same time.
- “Automatic Laser Control” cannot be combined with variable laser control via the **McBSP** interface (see **set_multi_mcbsp_in**).

General Notes

- Each individual contribution as well as the total correction cannot exceed a factor of 4.0 (clipping⁽¹⁾).
- If laser power and/or energy input into the to-be-processed material is not strictly proportional to the output values of the selected signal parameter, then **load_auto_laser_control** can be used to load a nonlinearity curve that defines this (application-specific) relationship, see **Section “Loading and Determining the Nonlinearity Curve”**, page 203.
- In addition, the selected signal parameter can neither exceed the value range allowed with **set_auto_laser_control** (parameter **MinValue** and **MaxValue**) nor the value range allowed for the respective output port. It is clipped correspondingly.
- For the laser control signal a corresponding default value is outputted (see **set_port_default** or **set_laser_off_default**), if:
 - Signals for “Laser Active” Operation are switched off when “Automatic Laser Control” is active
 - “Automatic Laser Control” itself is deactivated If no default value has been explicitly defined, the permitted maximum value is outputted. As substitute for the control parameters **HalfPeriod** and **PulseLength**, the parameter **Value** (the 100% value) from **set_auto_laser_control** is used.
- Once an “Automatic Laser Control” has been switched on with **set_auto_laser_control**, then the following can be changed subsequently by **set_auto_laser_params** or **set_auto_laser_params_list**:
 - the signal parameter
 - the 100% value
 - limit values assigned to it
 The **Mode** parameter cannot be changed subsequently.

(1) < DLL 612: overflow.

Position-Dependent Laser Control

To activate the position-dependent laser control for the output port specified by **set_auto_laser_control**, a user-defined scaling function must be loaded by **load_position_control**, see [Section "Notes on Loading a Scaling Function", page 198](#).

This scaling function represents a scaling factor as a function of the distance to the center of the **Image field**. After **load_program_file**, it is initialized for all distances with "factor 1.0". The "position-dependent" laser control is de facto deactivated by that. The table can be saved by **create_dat_file**.

When calculating the correction for "position-dependent laser control", the current Cartesian control values are used as a basis:

- including wobbel correction (#2 in [Chapter 7.3.6 "Output Values to the Scan System", page 180](#))
- including coordinate transformation in the virtual **Image field** (#3)
- including Processing-on-the-fly correction (#4)
- excluding head-specific coordinate transformation (#5)
- excluding **Image field** correction (#6)

Notes on Loading a Scaling Function

- For the **Scale(Position)** scaling function, **load_position_control** loads a table from an ASCII text file.
- The ASCII text file can contain one or several tables.⁽¹⁾
- Each table can contain up to 50 data points (**Position | Scale(Position)**).
- The **Scale(Position)** function is linearly interpolated from the data points.

For the scaling function tables, the following rules apply:

- Each table must begin with the line:
`[PositionCtrlTable<No>]`
`<No>` represents the table number.
- If the table contains multiple `[PositionCtrlTable<No>]` entries with the same `<No>`, then only the lines after the first entry are used. Only lines up to the next '[' character (that is not preceded by a semicolon) are used.
- Each data point (**Position | Scale(Position)**) is defined as follows:
`Position<n> = <Value>`
`Scale<n> = <Value>`
 where `<n>` corresponds to the index ($1 \leq <n> \leq 50$) of the data point. The values `<Value>` can be specified as (unsigned) floating point numbers. Decimal separator: period (.)
- If the table contains multiple data points with the same Index `<n>`, then the most recently read one is used.
- If the table contains multiple data points with the same position value **Position**, then the data point with the largest Index `<n>` is used. 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 percent of half the image-field side length.
 Example: $(X_{\max}|0)$ corresponds to 100%, $(X_{\max}|Y_{\max})$ corresponds to $2^{1/2} \times 100\%$.

(1) Even of another type, see [table 1, page 152](#).



- For $\langle\text{Value}\rangle$, the following ranges apply:
 $0.0 \leq \text{Position} \leq 150.0$ and
 $0.0 \leq \text{Scale}(\text{Position}) \leq 4.0$.
- Each instruction must be in a separate line.
- Spaces and tabs in a line (for example, between '=' and $\langle\text{Value}\rangle$) are ignored.
- Empty lines are ignored.
- Data points with invalid values are ignored.
- The data point of a particular index $\langle n \rangle$ is ignored if the corresponding $\text{Position}_{\langle n \rangle}$ and/or $\text{Scale}_{\langle n \rangle}$ 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, **load_position_control** has no effect (return value 1 or 13).
- If there is no entry for $\text{Position} = 0.0$, then an entry with $\text{Scale} = \text{Min}(\text{Scale}_{\langle i \rangle})$ is inserted (the smallest allowed value defined in the table is used for lower positions values). Likewise for $\text{Position} = 150.0$ with $\text{Max}(\text{Scale}_{\langle i \rangle})$.
- If the selected text file only contains a single valid data point with $\text{Scale}_{\langle n \rangle} = S$, then (for the entire position range) the scaling function $\text{Scale}(\text{Position}) = S$ is loaded. The correction has a multiplicative effect on the laser control signal. For $S = 1.0$, position-dependent correction is therefore switched off. Alternatively, this can also be achieved with **Name** = **NULL** in **load_position_control**.
- The table can be saved by **create_dat_file**.

Speed-Dependent Laser Control

When activating the “speed-dependent laser control” for the output port specified by `set_auto_laser_control`, the `Mode` parameter is used to select which input parameters are to be used for calculating the correction.

As reference value for the 100% speed, always the set (target) mark speed (set by `set_mark_speed` or `set_mark_speed_ctrl` applies (its change is never compensated by the “Automatic Laser Control”).

- `Mode` = 1 is intended (for consistency reasons) especially for analog scan systems. The current `Microstep` length per 10 µs is used as input. It may deviate from the target speed due to the 10 µs clock pulse rounding or with `[*]timed[*]` commands. Variations in the acceleration and deceleration phases cannot be compensated.
- `Mode` = 2 is intended as basic mode for `iDRIVE` scan systems⁽¹⁾. It can be combined with other special corrections (see below).
- Extensions to `Mode` = 2 (any combination possible):
 - +0 Basic mode for `intelliSCAN` systems
 - +4 Combines the speeds from `Mode` = 2 and `Mode` = 5 to a total speed. The 100% reference speed from `Mode` = 5 remains unconsidered. Instead, the encoder speeds are converted into galvanometer scanner bit speeds with the fly scaling factors and then vectorially added to the galvanometer scanner speed. A corresponding `Processing-on-the-fly` session must be active.

– +16

Basic mode for `excelliSCAN` scan heads

– +32

Position-dependent correction of the galvanometer scanner speed (in angular units) in a one for the `Image field` (“inverse speed correction table”)

Notes

- Usually `set_auto_laser_control`(`Mode` = 2) requires a special `intelliSCAN` firmware, which returns an actual speed corrected by the signal runtimes between the RTC6 PCIe Board and the scan head. If you have any questions as to whether your `intelliSCAN` is equipped with it or is upgradeable, contact SCANLAB.
- Usually a combination of “speed-dependent laser control” and a negative `LaserOn Delay` does not make sense.

(1) See Glossary entry on [page 26](#).

“Spot Distance Control”

The “Spot Distance Control” functionality is only available for scan systems equipped with **SCAN**ahead servo control (for example, excelliSCAN series).

With `set_auto_laser_control(Ctrl = 7)` (“Spot Distance Control”), temporal output pulses are controlled to a geometrically constant pulse distance. This pulse distance is defined by `spot_distance` or `spot_distance_ctrl`. The parameters `Value`, `MinValue` and `MaxValue` have no meaning with the “Spot Distance Control” functionality.

The speed correction occurs with `Mode = 2 + 16 + (optionally) 32`, (see above). “Spot Distance Control” cannot be combined with `Mode = 1, 2 or 5`.

The correction is more precise and dynamic than the comparable control with `Ctrl = 5` (HalfPeriod). For successful use, a laser with “pulse-on-demand” functionality should be used that outputs laser pulses immediately when triggered by the LASER1 signal. Frequency modulations are set with an accuracy of $1/64 \mu\text{s}$ for the pulse distance.

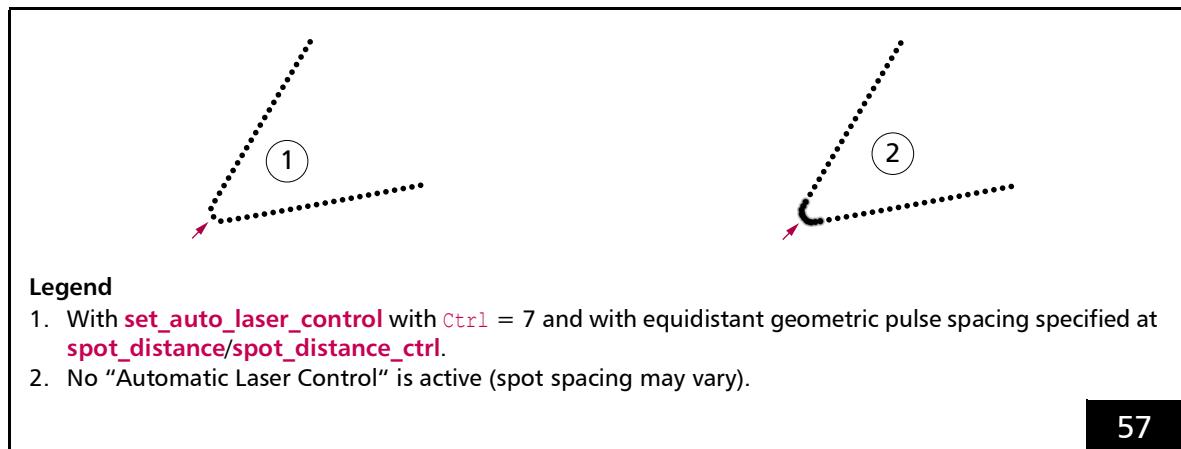
A comparison of a typical marking result with “Automatic Laser Control” (`Ctrl = 7`) and without “Automatic Laser Control” is shown in [Figure 57](#).

As an alternative to `set_auto_laser_control` with `Ctrl = 7`, the marking shown in [Figure 58](#) could also be carried out with **Sky Writing** switched on. However, when using `set_auto_laser_control` with `Ctrl = 7` the process time is often shorter.

Notes

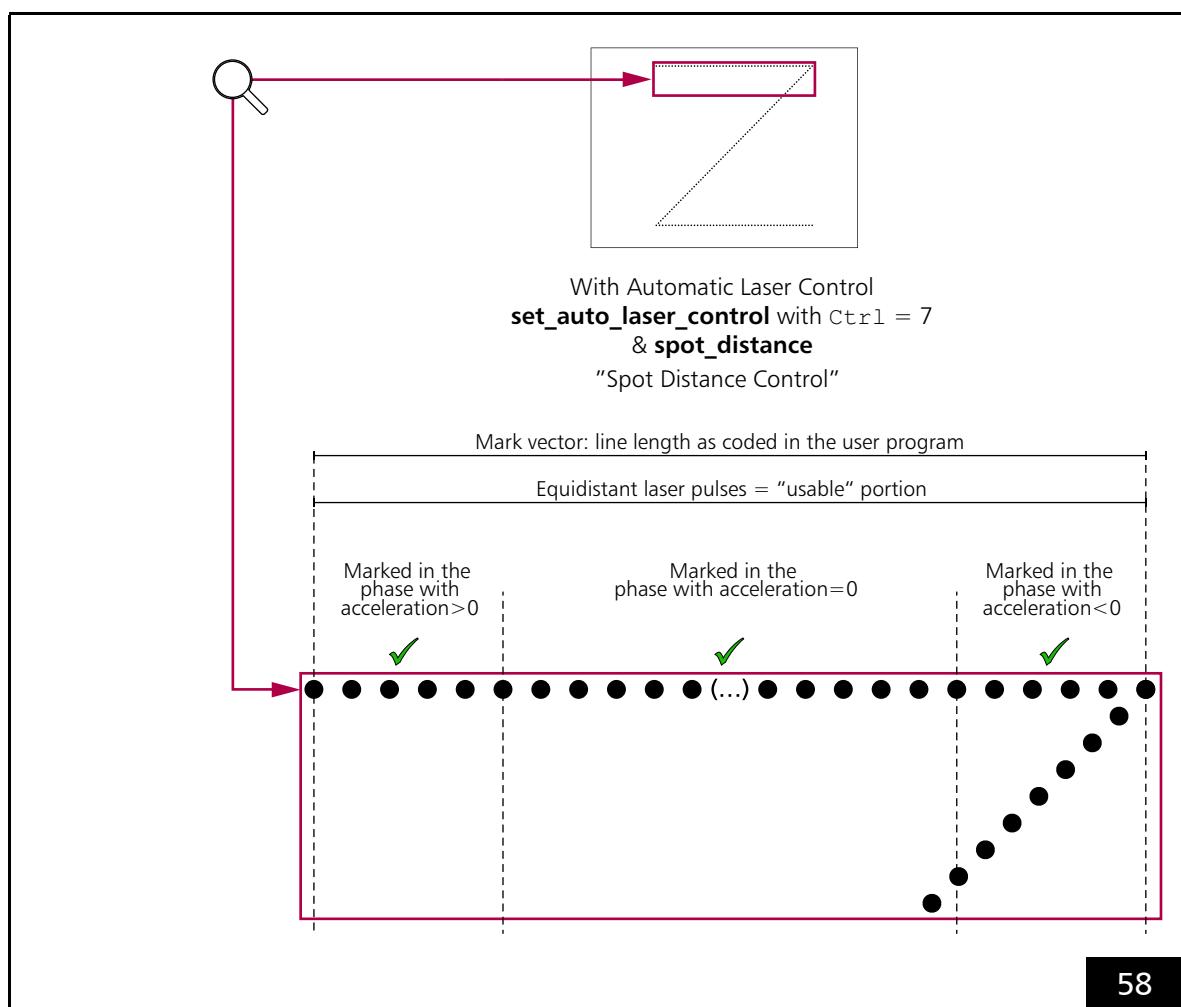
- `set_auto_laser_control(Ctrl = 7)` (“Spot Distance Control”) cannot be combined with:
 - “Vector-controlled laser control” by `set_vector_control(Ctrl = 5)` (HalfPeriod)
 - Laser Mode 4⁽¹⁾
 - Laser Mode 6⁽¹⁾
- `set_trigger/set_trigger4` (signal 24) records an internal control signal but *not* HalfPeriod.

(1) In Laser Mode 4 and Laser Mode 6, only a standby signal is outputted that cannot be varied.



57

Example marking result: "sharp" corner.



58

For `set_auto_laser_control` with `Ctrl = 7` the marking result shows *largely* equidistant spots as long as the acceleration does not change too much within a $10 \mu\text{s}$ cycle.

Encoder-Speed-Dependent Laser Control

`set_auto_laser_control(Mode = 5)` is intended for a pure encoder speed-dependent correction, if only the workpiece moves and the galvanometer scanners (ideally) are idle.

The 100% reference speed is defined by `set_encoder_speed_ctrl` or `set_encoder_speed`. Processing-on-the-fly should not be active at the same time.

For a combination of galvanometer scanner speeds and encoder speeds in a

`Processing-on-the-fly session`, see `Mode = 6 = 2 + 4`.

Loading and Determining the Nonlinearity Curve

- For the `Scale(Percent)` nonlinearity curve, `load_auto_laser_control` loads a table from an ASCII text file.
- The ASCII text file can contain one or several tables.⁽¹⁾
- Each table can contain up to 50 data points (`Percent` | `Scale(Percent)`).
- The `Scale(Percent)` function is linearly interpolated from the data points.

For the tables, the following rules apply:

- Each table must begin with the line:
`[AutoLaserCtrlTable<No>]`
`<No>` represents the table number.
- If the table contains several
`[AutoLaserCtrlTable<No>]` entries with the same
`<No>`, then only the lines after the first entry are used. Only lines up to the next '[' character (that is not preceded by a semicolon) are used.
- Each data point (`Percent` | `Scale(Percent)`) is defined as follows:
`Percent<n> = <Value>`
`Scale<n> = <Value>`
 where `<n>` corresponds to the index ($1 \leq <n> \leq 50$) of the data point. The values `<Value>` can be specified as (unsigned) floating point numbers.
 Decimal separator: period (.)
- If the table contains multiple data points with the same `Index <n>`, then the most recently read one is used.
- If the table contains multiple data points with the same percent value `Percent`, then the data point with the largest `Index <n>` is used. Equality is checked to within $\pm 0.01^\circ$.

(1) Even of another type, see table 1, page 152.

- The percent value is relative to the 100% value from **set_auto_laser_control** (Parameter **Value**) or dynamically from a ““vector-controlled laser control””, see **Section “Vector-Defined Laser Control”**, page 205.

In the following example, a nonlinearity factor of 1.2 is set for a 1.5x multiple of the target value:

Percent<n> = 150

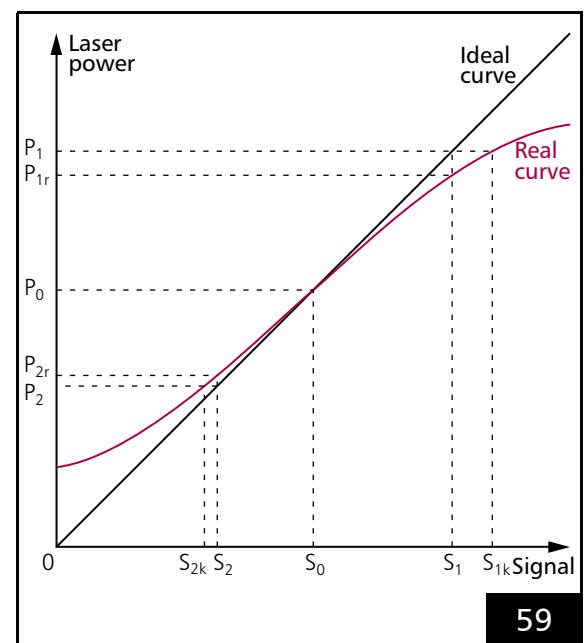
Scale<n> = 1.2

- 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 (for example, 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 **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>)**.

After **load_program_file** this function is initialized for all percentage values with “Factor 1.0”, the nonlinear laser control is “deactivated”. Alternatively, this can also be achieved with **Name = NULL** in **load_auto_laser_control**.

The table can be saved by **create_dat_file**.

The example diagram in **Figure 59** illustrates how the nonlinearity curve can be determined.



Laser power progression – example of determining a nonlinearity curve.

The straight line in the diagram describes an ideal 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 simulates a realistic relationship.

S_0 is the signal parameter value defined as the target value and P_0 is the associated laser power. At point $(S_0 | P_0)$ (this corresponds to the data point $\text{Percent}_0 = 100, \text{Scale}_0 = 1.0$) the two curves are normalized to each other. The combination of a nonlinearity curve with a ““vector-controlled laser control”” is therefore generally not recommended.

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. The following two value pairs are then to be entered as data points for the nonlinearity curve:

$$\text{Percent1} = S_1/S_0 \times 100 = P_1/P_0 \times 100$$

$$\text{Scale1} = S_{1k}/S_1$$

$$\text{Percent2} = S_2/S_0 \times 100 = P_2/P_0 \times 100$$

$$\text{Scale2} = S_{2k}/S_2$$



Vector-Defined Laser Control

The “vector-controlled laser control” allows a signal parameter to be varied linearly along a mark vector or jump vector.

To initialize the ““vector-controlled laser control””, **set_vector_control** must be used to specify which signal parameter is to be varied with which initial value (parameters **Ctrl** and **Value**).

Then the signal parameter is varied linearly along a parameterized mark or jump vector. The end value is automatically the start value for a subsequent **[*]para[*] Command**.

Notes

- List commands for explicitly changing the signal parameter output value are temporarily effective or not at all.
- **[*]para[*] Command**s always use the end value of the previous **[*]para[*] Command** as their start value.
Control commands that write to the same output port should be avoided while processing a list of **[*]para[*] Command**s.
- If the same output port is selected via **set_auto_laser_control** for **Ctrl**, the control parameter from the **[*]para[*] Command**s acts as 100% value for the laser control.
Special care should be taken with **set_vector_control** (**Ctrl** = 7) (Defocus): The setting of the signal value is always done immediately. This can lead to **Hard jumps** on the varioSCAN.
- During execution of **[*]para[*] Command**s, the **Sky Writing** mode, see [Chapter 7.2.4 “Sky Writing”, page 159](#), is *not* taken into account (but also not deactivated).

7.4.10 Synchronization of the RTC6 Clock Cycle and an External Clock Signal

The laser pulse signals of a free-running laser and the laser control of the RTC6 PCIe Board can be synchronized. Non-flush line starts are thereby avoided, see [Figure 60](#).

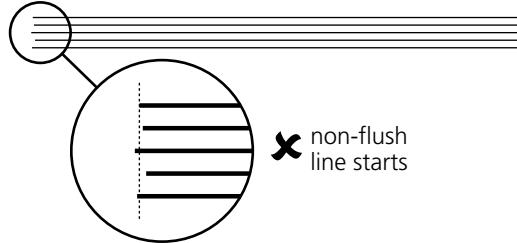
Here, the RTC6 PCIe Board accepts the external clock signal as the master clock. It must be supplied at the digital input port DIGITAL IN1 of the [LASER Connector](#).

The synchronization is switched on and off by [Bit #6](#) of [set_laser_control](#). By [Bit #5](#), it can be set whether to use the rising or falling edge of the external clock for synchronization. This automatically synchronizes the RTC6-internal laser control signals to the external clock signals.

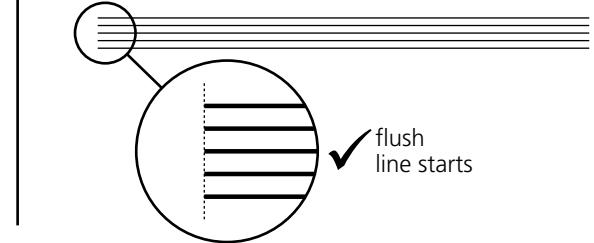
Notes

- The frequency of the master clock must meet the following requirements:
 $f = k \times 100 \text{ kHz}$ ($k \leq 64$) and
 $f = 64 \text{ MHz} / n$ (k and n integer).
Therefore, only the following frequencies can be used: 100 kHz, 200 kHz, 400 kHz, 500 kHz, 800 kHz, 1 MHz, 1.6 MHz, 2 MHz, 3.2 MHz, 4 MHz, 6.4 MHz. The allowed external frequency deviation from an integer multiple of 100 kHz is $\pm 15.625 \text{ ns}$ per $10 \mu\text{s}$.
- The supplied clock signal must be a TTL signal. The minimum pulse length or pulse pause of the clock signal should be 80 ns. See also [Chapter 4.6.1 "LASER Connector", page 72](#), [Section "2-Bit Digital Input Port", page 73](#).
- No synchronization is carried-out, if the synchronization is activated but there is no valid clock signal at DIGITAL IN1 of the LASER-connector. In this case the RTC6 PCIe Board uses its internal $10 \mu\text{s}$ clock cycle.
- If the synchronization is activated by [set_laser_control](#) by setting [Bit #6](#), then the synchronization of RTC6 PCIe Board and external clock is subtly (depending on the respective phase position within 1.2 ms at most).

RTC6 10 μs clock period and laser pulses of the free-running laser are *not* synchronized:



RTC6 10 μs clock period and laser pulses of the free-running laser are synchronized:



Free-running laser and RTC6 PCIe Board – example of marked lines.

60

7.4.11 Pulse Synchronization Mode

The **Pulse Synchronization Mode**:

- Delays the output of a LASER1 pulse until a new clock signal pulse of the laser at DIGITAL IN1 pin⁽¹⁾ is detected, see also [Figure 61](#).
- Is enabled by `set_laser_pulse_sync(Mode > 0)`
- Is disabled by `set_laser_pulse_sync(Mode = 0)`

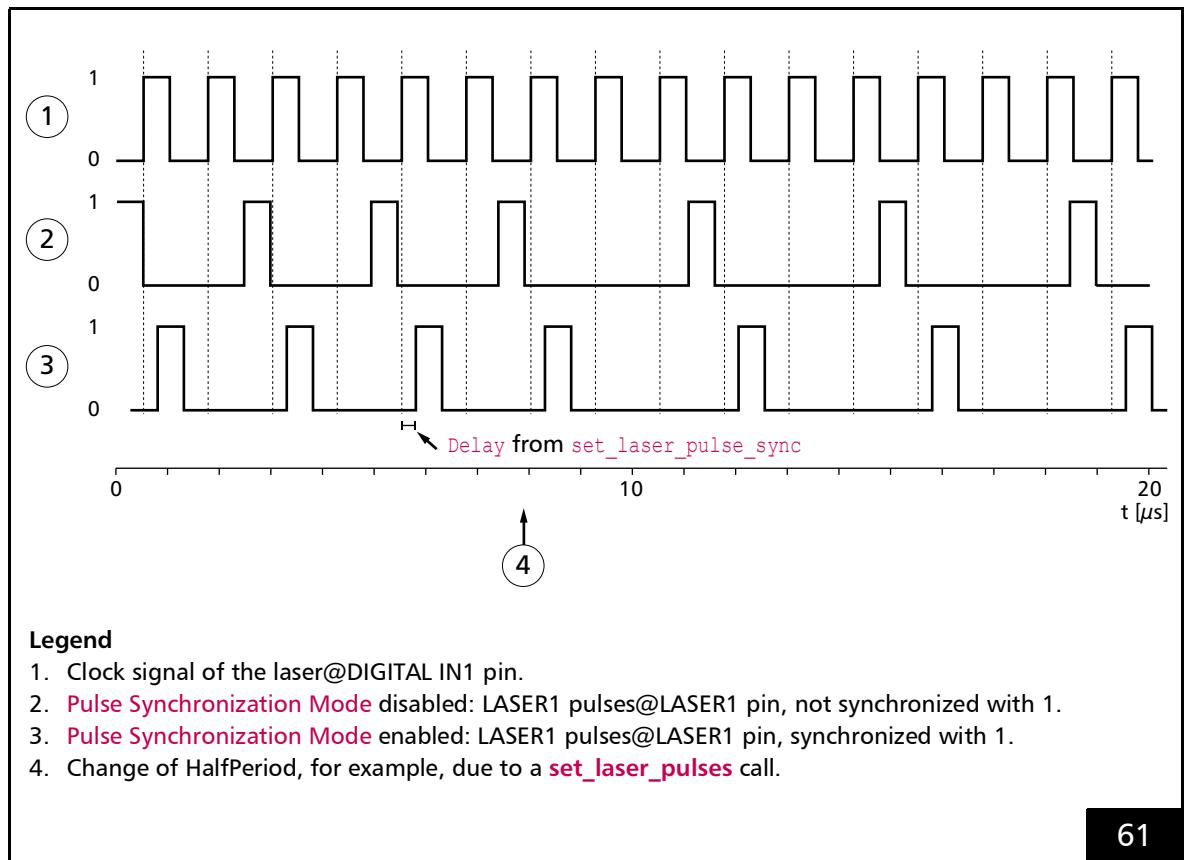
Observe the following when selecting the parameter values:

- By `set_laser_pulse_sync(Delay)`, the LASER1 signal can be delayed by a fixed value. The `Delay` value must be \leq the period set by
 - `set_laser_pulses`
 - `set_laser_timing`
 - `set_laser_pulses_ctrl`
 - `set_auto_laser_control`
- For a `Delay` value $\geq (2 \times \text{HalfPeriod}$ from `set_laser_pulses`), the following applies:
 - The first LASER1 pulse is delayed by the `Delay` value
 - The subsequent LASER1 pulse is ignored
- The LASER1 signal pulse length remains unchanged and depends on the preceding configuration (for example, by `set_laser_pulses`).

Notes

- The supplied clock signal must be a TTL signal. The minimum pulse length or pulse pause of the clock signal should be 80 ns. See also [Chapter 4.6.1 "LASER Connector", page 72](#), [Section "2-Bit Digital Input Port", page 73](#).
- By Bit #5 of `set_laser_control`, it can be set whether to use the rising or falling edge of the external clock for synchronization.
- **Pulse Synchronization Mode** can be combined with "Spot Distance Control" as well as with the other variants of "Automatic Laser Control" such as `PulseLength` or `HalfPeriod`, see [Chapter 7.4.9 "Automatic Laser Control", page 196](#).
- **Pulse Synchronization Mode** can be combined with [Laser Mode 4](#) or [Laser Mode 6](#).
- **Pulse Synchronization Mode** *cannot* be combined with laser modes that use LASER2 signal (CO_2 Mode, Pulse Picking Laser Mode). **Pulse Synchronization Mode** because:
 - It only affects the LASER1 signal
 - It does not change the LASERON signal
 - It does not change the LASER2 signal
- **Pulse Synchronization Mode** *should not* be combined with cycle synchronization, see [Chapter 7.4.10 "Synchronization of the RTC6 Clock Cycle and an External Clock Signal", page 206](#).

(1) At the [LASER Connector](#), see [Figure 17](#).



7.5 Marking Dates, Times and Serial Numbers

The **RTC6 DLL** contains a series of commands to mark the current time, current date or the serial number of products.

Before times, dates and serial numbers can be marked, the required characters and text strings must be defined as indexed characters and indexed text strings.

Separate text strings can be defined for marking times/dates and serial numbers. See [Section "Defining Indexed Text Strings for Time, Date and Serial Number", page 118](#).

7.5.1 Marking the Time and Date

By **time_update** the PC time/date is transferred to the RTC6 PCIe Board. This is necessary after every PC restart. After that, the board (as long as it remains energized) internally counts the date/time with the quartz-controlled 10 μ s clock to the second.

By **time_fix**, **time_fix_f** or **time_fix_f_off** the current time/date of the board is held in a cache.

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**.

To mark the date and time, the Gregorian or Julian date can be set as well as the 12- or 24-hour format.

For marking dates of expiry, **time_fix_f_off** is available to fix a forward date based on the current date and current time.

7.5.2 Marking Serial Numbers

By **mark_serial** and **mark_serial_abs**, up to 12-digit serial numbers can be marked. It can be specified how leading zeros are handled.

The RTC6 PCIe Board manages up to 4 serial-number-sets (each with its own serial number and increment size). Serial-number-set 0 is selected at initialization with **load_program_file**.

Other serial-number-sets must be selected in advance by **select_serial_set** or **select_serial_set_list** (see notes below).

By **set_serial**, **set_serial_step** or **set_serial_step_list**, a starting serial number (max. 10 digits) and an increment size for each serial-number-set can be specified. At initialization with **load_program_file** all starting serial numbers are set to 0 and all increment sizes are set to 1.

Each call of **mark_serial** or **mark_serial_abs** causes the current serial number to be incremented (yet before execution of the serial number marking) by the specified increment size.

If a serial number is to be omitted a blank marking can be executed (**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**.
- 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. This also lets you determine (among other things) whether the current number has been or has not been incremented after an uncontinued aborted list.
- 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** (Note that the RTC6 PCIe Board does not prohibit 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`.

- `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).
- `set_max_counts` allows specification of the maximum number of `External Starts` and thus the maximum number of externally started markings. The number of already occurred `External Starts` can be obtained with `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 Control and Laser Control

8.1 iDRIVE Functions

SCANLAB iDRIVE scan systems⁽¹⁾ utilize the iDRIVE technology. This 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 RTC-control board facilitates support of all the new features (see also the following section).

This allows users to adjust a number of settings of the respective scan system, for example,

- To select which data it has to transmit to the RTC-control board
- To choose from different dynamic settings (tunings)
- To set the **PosAck** limit value
- To set the effective calibration of the scan system
- To set the start behavior of the scan system
- To perform a fault diagnosis
- To perform a functional test of the data transfer

For more information, refer to the manual of the scan system.

iDRIVE functions are executed by **control_command**⁽²⁾.

8.1.1 Transfer Protocol

Data transfer between RTC6 PCIe Board and scan system is carried out according SL2-100 protocol.

The SL2-100 protocol supports the full functionality of iDRIVE technology.

With iDRIVE scan systems⁽¹⁾ this protocol allows, for example, status signals of the x axis and y axis to be separately and simultaneously evaluated. For a 3D scan system, z axis status signals can be simultaneously read back by the channel of the scan head connector to which the z axis is connected.

The **XY2-100 Converter (Accessory)** can be used with iDRIVE scan systems⁽¹⁾ equipped with an interface for XY2-100 protocol or XY2-100 Enhanced protocol.

(1) See Glossary entry on [page 26](#).

(2) See also [Section "Folder iSCANCfg", page 31](#).

8.1.2 Configuring the Data Signal Transmission Behavior of the Scan System

Setting Data Types

The digital servo architecture of iDRIVE scan systems⁽¹⁾ allows a wide variety of data signals to be returned from the scan system to the RTC-control board.

Each axis has its own status channel on which data is transmitted to the RTC-control board every 10 µs:

- STATUS channel
 - Designed for the x axis
(Galvanometer scanner 2)
- STATUS1 channel
 - Designed for the y axis
(Galvanometer scanner 1)

This opens up possibilities such as monitoring the actual values of the galvanometer scanners during an application or carrying out comprehensive troubleshooting in case of operational malfunction.

`control_command(Data = 05nnH)` allows to set which data the scan system has to transmit to the RTC control board. The available data types are described in detail in the manual of the respective scan system and (in parts) in the command reference of `control_command`. The set data source is transferred until another data source is set.

After every power-up or reset (after the initialization has been completed), the scan system transmits (on all receive channels) the `XY2-100 status word`.

Reading Out Data

At any time, data received by the RTC6 PCIe Board can be:

- Read out asynchronously by `get_value`, `get_values` or `get_head_status`
- Synchronously recorded by `set_trigger`/`set_trigger4`

See also [Chapter 7.3.7 "Status Monitoring and Diagnostics", page 182](#).

Note that switching of the data source is followed by a short (serial transmission-related) delay of typically 50 µs before the first data is transmitted, see also comment at [control_command, page 946](#).

The value ranges of these data and the possible status states are described in [Chapter 20 "Appendix E: iDRIVE Scan Systems – Control Commands and Signals Transmitted to RTC Control Boards", page 916](#).

Notes

- `get_head_status` queries the `XY2-100 status word`. With SL2-100 protocol-compliant data transfer, the scan system always transfers the `XY2-100 status word` in parallel with other status information.

Important: If the `XY2-100 Converter (Accessory)` is used for control, then it must explicitly be set that the scan system has to transfer the `XY2-100 status word` to the RTC6 PCIe Board. Otherwise, `get_head_status` returns unusable values.

(1) See Glossary entry on [page 26](#).

8.1.3 Monitoring the Positioning

For some applications, it is important to monitor and, if necessary, to document the scan system positioning even during operation.

For this purpose, the actual position of the scan axes must be set to be returned from the scan system by **control_command**. The returned actual position values can be queried then by **get_values** or recorded by **set_trigger/set_trigger4**⁽¹⁾.

If the returned actual positions of the scan axes are to be compared with the Cartesian target coordinate values (X, Y, Z), then these must be transformed back by the corrections made on the RTC6 PCIe Board, [Chapter 7.3.6 "Output Values to the Scan System", page 180](#).

For runtime reasons, the backward transformation needs to subsequently be performed by the PC rather than on the RTC6 PCIe Board itself. For this, all correction and transformation settings currently assigned to the scan system can be transferred from the RTC6 PCIe Board to the PC by **upload_transform**. Afterwards, an individual xy value pair or an individual z value can be backward transformed by **transform**. With **get_transform** (see also **get_waveform**), an entire series of xy value pairs or z values previously recorded by **set_trigger/set_trigger4** can be backward transformed.

Notes

- For some forward transformations, a backward transformation is not possible:

Forward transformation	Backward transformation possible?
Wobbel motion (#2 on page 180)	no
Global coordinate transformation (#3 on page 180)	no
Processing-on-the-fly correction (#4 on page 180)	no
Coordinate transformations (total matrix and offset) to the x and y coordinates (#5 and #6 on page 180)	yes
Offset to the z coordinate (#6 on page 180)	yes
Clipping to the limits of the controllable Image field (#7 on page 180)	no
2D Image field correction / 3D image field correction (#8 on page 180)	yes
Gain and offset correction of automatic self-calibration (#9 on page 180)	yes
Clipping to the maximum possible control values	no

- Furthermore, backward transformation is not possible if a noninvertable transformation matrix has been defined during forward transformation (notice: clipping to ± 50 for each individual matrix coefficient; see [Chapter 8.2 "Coordinate Transformations", page 223](#)). The non-invertability is already reported by **upload_transform**.

(1) Due to communication runtimes, the currently returned actual positions are several clock pulses later than the currently outputted control signals.

- 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.
- By **get_values**, 4 arbitrary signals can be queried at the same time. Example:
 - the actual position of galvanometer scanner 2 by StatusAX
 - actual position of galvanometer scanner 1 by StatusAY
 - the actual z axis position by StatusBX
 - an additional desired signal, for example, LaserOn
 In contrast, **get_value** (not: **get_values**) 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** 4 signals by 4 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 or Z for backward transformation.
- **transform** and **get_transform** also allow specification of which partial transformations are to be performed.
- Values queried by **get_values**, or arbitrary synthetic values can be backward transformed by **transform**. During backward transformation of synthetic values beyond the forward transformation's achievable **Image field**, values sometimes are only calculated by extrapolation, due to possible range exceedances 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 RTC6 PCIe 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 marked pattern. But you can check if the scan system correctly processed the RTC6 PCIe Board 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 Selecting the Tuning (Dynamics Setting)

SCANLAB can optimize the dynamics setting of scan systems (tuning) to accommodate differing requirements of diverse applications regarding the laser positioning dynamics, for example:

- Vector tuning
 - to execute vectors or circular arcs at a constant processing speed
- Jump tuning
 - to execute jumps of minimized duration

iDRIVE scan systems⁽¹⁾ can be optionally equipped with several tunings. For different applications, the suitable tuning can be switched to – separately for each axis – by **control_command**(`Data = 11nnH`).

For scan systems equipped with one or several **Jump tunings**, you can also activate **Jump Mode** (and hereby tuning autoswitching) for 2D jumps, see [Chapter 8.1.5 "Jump Mode", page 216](#).

The default set start behavior is that the scan system starts with tuning number 0 upon power-up or after a reset.

(1) See Glossary entry on [page 26](#).

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".

This requires the scan system to be equipped with a **Jump tuning**, see also [Section "Requirements and Activation", page 217](#).

Functional Principle

In the default setting (after `load_program_file`), both **Jump commands** and **Mark commands** are executed in vector mode:

- The jump length gets subdivided into individually executable **Microsteps** in accordance with the current jump speed. If the scan system is only equipped with a **Jump tuning**, then the **Microsteps** execute using this tuning.
- A **Jump Delay** defined by `set_scanner_delays` is executed before a subsequent list command.

In contrast, when **Jump Mode** is enabled and activated by `set_jump_mode` or `set_jump_mode_list`, every 2D jump (see below) is executed as follows:

- The entire jump length of the 2D jump is controlled as a "Hard jump" over a time dimensioned jump of 10μ duration. The target position is executed without **Microstepping**.
- The jump executes with a **Jump tuning**. `set_jump_mode` can be used to designate which **Jump tuning** to use. If a different tuning has been set before the jump, then the RTC6 PCIe Board automatically switches at the beginning of the jump to the tuning specified by `set_jump_mode`.
- At the end of the 2D jump, the RTC6 PCIe Board automatically switches to a **Vector tuning** (if the scan system is equipped with one and if a corresponding setting has been 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 [Section "Jump-Length-Dependent Jump Delays", page 217](#). 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** has been 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 10μ clock cycle. 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 (split-up into **Microsteps** and without further servo autoswitching). Here, the **Vector tuning** is used that has been most recently set at the end of **Jump Mode**, unless deactivation has been 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, intellilcube, intelliWELD or intelliDRILL scan system.
- As of scan system firmware ≥ 2078 .
- 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 in the scan system 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** has been not already enabled.

Jump-Length-Dependent Jump Delays

When executing a “**Hard jump**”, it takes the scan head some time to reach the specified position.

The RTC6 PCIe Board takes this delay (also called step response) into account by appending a **Jump Delay** at the end of the jump.

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 RTC6 PCIe Board in a user-specific **Jump Delay** table. With **Jump Mode** enabled, the RTC6 PCIe Board 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 (that is, 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 **load_jump_table_offset**.

Alternatively, the **Jump Delay** table can also be automatically determined by

load_jump_table_offset (parameter **Name** = **NULL**).

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.

When the **Jump Delay** table has been loaded and a new **RTC6DAT.dat** file is created by **create_dat_file**, then this table is automatically loaded upon the next **load_program_file**.

Determining **Jump Delay** Values Experimentally

The user manual of the scan system 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**. 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 in the ASCII text file.

Notes on Loading Determined **Jump Delay** Values

- For jump length values and **Jump Delay** values, **load_jump_table_offset** loads a table from an ASCII text file.
- The ASCII text file can contain one or several tables.⁽¹⁾
- Each table can contain up to 50 data points (*Length* | *Delay*(*Length*)).
- The complete (internal) **Jump Delay** table *Delay*(*Length*) is linearly interpolated from the data points.

For the tables, the following rules apply:

- Each table must begin with the line:
`[JumpTable<No>]`
`<No>` represents the table number.
- If the table contains multiple `[JumpTable<No>]` entries with the same `<No>`, then only the lines after the first entry are used. Only lines up to the next '`'` character (that is not preceded by a semicolon) are used.
- Each data point (*Length* | *Delay*(*Length*)) is defined as follows:
`Length<n> = <LengthValue>`
`Delay<n> = <DelayValue>`
 where `<n>` is the data point index ($1 \leq <n> \leq 50$).
 The `<Value>` numbers can be supplied as (unsigned) floating point numbers. Decimal separator: period (.).
- 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 .

(1) Even of another type, see table 1, page 152.



- For $\langle\text{Value}\rangle$ the following ranges apply:
 $0.0 \leq \text{Length} \leq 1048576.0$
 $0.0 \leq \text{Delay}(\text{Length}) \leq 65535.0$
Delay values are supplied in units of $10 \mu\text{s}$,
jump lengths in bits.
- Each instruction must be in a separate line.
- Space characters and tabs within a line (for example, between '=' and $\langle\text{Value}\rangle$) are ignored.
- Empty lines are ignored.
- Data points with invalid values are ignored.
- The data point of a particular index $\langle n \rangle$ is ignored if the corresponding $\text{Length}\langle n \rangle$ and/or $\text{Delay}\langle n \rangle$ 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 **load_jump_table_offset** has no effect (return value 1 or 13).
- If there is no entry of $\text{Length} = 0.0$, then one with $\text{Delay} = \text{Min}(\text{Delay}\langle i \rangle)$ is inserted (the smallest valid value encountered is filled downward). The same applies to $\text{Length} = 524288.0$ with $\text{Max}(\text{Delay}\langle i \rangle)$.
- If the specified text file contains only one valid data point with $\text{Delay}\langle n \rangle = D$, then the **Jump Delay** table $\text{Delay}(\text{Length}) = D$ (for the whole jump length range) is loaded.

Automatic Determination of the **Jump Delay** Table

You can initiate automatic determination of the **Jump Delay** table by **load_jump_table_offset** (parameter **Name** = **NULL**) if you had previously enabled and activated **Jump Mode** successfully by **set_jump_mode** (**Flag** = 1).

For automatic determination, "Automatic Laser Control" is deactivated, the data type to be 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 restored when **load_jump_table_offset** 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** (not: **set_trigger4**) and retrieved by **get_waveform**. The data is analyzed for the timepoint at which the specified position tolerance (**PosAck** parameter) has been last exceeded, that is, 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]. **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, 4 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 only whole multiples of 16 are returned with an **XY2-100 Converter (Accessory)**. 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...65,535). The Offset can be used to compensate for measurement runtime latencies (for example, caused by an **XY2-100 Converter (Accessory)**, by tuning switching or by a runtime latency of the signal returned by the scan system) 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.
- `load_jump_table` and `load_jump_table_offset(Offset = 0)` are identical.
- 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 **PosAck** Limit Value

`control_command(Data = 15nnH)` can be used to set the **PosAck** limit value nn. The default start behavior is for the scan system to set the limit value to 0.28% of the full position range after every power-up.

If other limit values are desired, they must be separately set for each axis.

8.1.7 Configuring the Effective Calibration

The servo electronic can be configured by `control_command(Data = 12nnH)` to down scale the position values received from the RTC6 PCIe Board by a specific factor (1, 1/2, 1/4 or 1/8). The position signals (optionally) returned by the scan systems to the RTC6 PCIe Board 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 scaling factor of 1 upon power-up.

By `control_command(Data = 053FH)`, the currently set scaling factor can be read out.

8.1.8 Configuring the Start Behavior

The default configuration of iDRIVE scan systems⁽¹⁾ is set as follows:

- Tuning number 0, see also [Chapter 8.1.4 "Selecting the Tuning \(Dynamics Setting\)", page 215](#)
- PosAck limit value is B8_H (corresponds to 0.28% of the full position range of 2¹⁶ bit), see also [Chapter 8.1.6 "Configuring the PosAck Limit Value", page 221](#)
- Scaling factor = 1, see also [Chapter 8.1.7 "Configuring the Effective Calibration", page 221](#)

These settings can be changed by **control_command**. The changed settings are only temporary, however they can be additionally saved as starting settings for subsequent power-ups by **control_command**(**Data** = 0A00_H).

The return behavior of the scan system can only be temporarily changed. After a power-up, the scan system transmits the XY2-100 status word, see also [Section "Configuring the Data Signal Transmission Behavior of the Scan System", page 212](#).

8.1.9 Fault Diagnosis and Functional Test

If a problem occurs, the versatile status return functions of the iDRIVE scan system⁽²⁾ can be used for scan system diagnosis, too.

For example, an event code can be queried that indicates which event has been responsible for the change to an error state.

To verify that data transfer capability between the RTC6 PCIe Board and a scan system is intact, by **control_command**(**Data** = 21nn_H) an 8-bit value nn – separately for each axis – can be transmitted 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.

This 20-bit value is returned until **control_command**(**Data** = 05nn_H) is used to select another data type to be returned by the scan system, see [Section "Configuring the Data Signal Transmission Behavior of the Scan System", page 212](#).

Prior to a transfer test, the data type currently selected for transmission can be cached by **control_command**(**Data** = 17FF_H). After the test, **control_command**(**Data** = 1700_H) restores the same transfer behavior as before the test.

(1) See Glossary entry on [page 26](#).

(2) See Glossary entry on [page 26](#).

8.2 Coordinate Transformations

For precise set-up of the scan system relative to the **Image field** (or, if the **Option "Second Scan Head Control"** is enabled, two scan heads can be adjusted relative to a **common Image field**), a linear coordinate transformation can be defined (separately for the first and second scan head connectors) for all x and y output coordinates (x|y) defined by **Vector commands** or **"Arc" commands**:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = M \times \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$$

The (2 x 2) total matrix M is thereby automatically calculated by the RTC6 PCIe Board as a product of a scaling matrix M_S , a rotation matrix M_R and a general transformation matrix M_T :

$$M = M_T \times M_R \times M_S$$

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 first and second 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 (in accordance with mathematical convention: 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 :

$$M_T = \begin{bmatrix} \cos \alpha & -\sin \alpha \\ \sin \alpha & \cos \alpha \end{bmatrix}$$

Example:

`set_matrix(1, 0.5, -0.866, 0.866, 0.5, 1)`
 defines a rotation by 60° (counterclockwise)
 around the center of the **Image field** for the first
 scan head connector.
 This can also be achieved by `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 RTC6 PCIe Board calculates the total matrix M independently of the order in which the 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 take place exclusively around the centerpoint of the **Image field**; mirroring is relative to the axes.
- 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 accumulated and cached (`at_once = 0` or `2`). The most recently called `at_once` parameter value determines when a (accumulated) transformation takes effect.

- Before the total transformation is applied to the current position, the **Signals for "Laser Active" Operation** are switched off with `at_once = 0...2`. They remain unchanged with `at_once = 3`.
- With `at_once = 1` or `3`, all settings (only) accumulated until then are processed immediately and simultaneously. In the process, the scan system axes are moved from the current position to the corrected position at the defined jump speed. Consequently, this can require some clock cycles.
This needs to be observed especially when RTC6 PCIe Boards are master/slave synchronized and are supposed to execute different jump lengths. With this use case, the coordinate transformations should be executed prior the synchronized start with `at_once = 1` and their end should be waited for.
Any **Scanner Delays** are not initialized. The **INTERNAL-BUSY list execution status** is set while the jump to the corrected position is executed.
- With `at_once = 2`, the accumulated settings only become effective upon execution of the next **jump_abs**, **jump_rel**, **goto_xy** or **goto_xyz** (but not other **Jump commands** such as **jump_abs_3d** or **jump_rel_3d**) unless afterwards another call triggers immediate execution. Correction of the current output position then occurs together with the specified coordinate jump. This eliminates unnecessary galvanometer scanner motions (incl. delays).
Example: The following command sequence 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 );
```

Notes on data recording:

- The galvanometer scanner movement of **jump_abs** or **jump_rel** at the beginning of the list can be recorded with **set_trigger** (signal type **7, 8, 9**), if no coordinate transformation with `at_once = 2` has been applied before.
- In case of movements *after* a coordinate transformation (for example, by **set_offset_list** with `at_once = 2`) the "sample values" (**set_trigger** signal type **7, 8, 9**) are immediately set to the target coordinates and remain unchanged for the duration of the movement. The movement recorded here does not appear like **Microstepping**⁽¹⁾ has been performed, but rather like a **Hard jump**. The real galvanometer scanner movement can be recorded with **set_trigger**, signal type **25** and **26** (transformed control values).
- In case of movements caused by coordinate transformation in a list (for example, by **set_offset_list** with `at_once = 1`), the "sample values" (**set_trigger** signal type **7, 8, 9**) remain unchanged for the duration of the movement. The real galvanometer scanner movement can be recorded with **set_trigger**, signal type **25** and **26** (transformed control values).
- Settings via control commands by `at_once = 0` are only saved as long as no list is running. When the execution is started or the list is already running, the settings take effect immediately before the next list command.
Settings via list commands with `at_once = 0` are only saved until they are retrieved elsewhere (`at_once > 0` or by a control command).
- If no correction table is assigned to the corresponding scan head connector, then the new settings for the coordinate transformations are only stored on the RTC6 PCIe Board. They take effect when a correction table is assigned.

(1) See [Chapter 7.1.2 "Microstepping", page 139](#).

- Coordinate transformations are applied to all to-be-outputted coordinates from all **Vector commands** (`[*]jump[*]` or `[*]mark[*]`) list commands, but also **goto_xy** or **goto_xyz** and **"Arc"** commands. See also [Chapter 7.3.6 "Output Values to the Scan System", page 180](#).
 - With a scaling matrix, the effective jump speed and mark speed changes.
 - With 3D vector commands:
 - The transformation matrix only affects the x and y components
 - The offset affects all three components
 - In order to utilize the complete real **Image field** with coordinate transformations (such as rotations, shrinkages or shifts), the extended value range of the virtual **Image field** can be used even without Processing-on-the-fly, see [Chapter 7.3.3 "Virtual Image Field", page 168](#).
 - Coordinate transformations for the virtual **Image field** can be defined, see [Section "Coordinate Transformations in the Virtual Image Field", page 168](#).
- See also [4](#) in [Chapter 7.3.6 "Output Values to the Scan System", page 180](#).

Notes for RTC4 Users

- With RTC6 PCIe Boards, coordinate transformations can no longer be set upon loading a correction file by **load_correction_file**. Instead, coordinate transformations are separately defined for the first and second scan head connector and serve the same purpose as those of **load_correction_file** of the RTC4.
- RTC6 PCIe Boards do not support the coordinate transformations (collectively for both *scan heads*) before **Microstepping** which is possible with the RTC4. The global coordinate transformations have a slightly different effect than the coordinate transformations above, see [3](#), [5](#) and [6](#) in [Chapter 7.3.6 "Output Values to the Scan System", page 180](#).

8.3 Online Positioning

The preceding [Chapter 8.2 "Coordinate Transformations", page 223](#) details how to precisely align a scan system relative to the **Image field**, see also [Section "Coordinate Transformations in the Virtual Image Field", page 168](#).

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 $10 \mu\text{s}$ clock of the RTC6 PCIe Board) while positioning a workpiece and aligning the scan system by the control commands described in [Chapter 8.2 "Coordinate Transformations", page 223](#). For applications in which such timing is important, commands for so-called **Online Positioning** are available. Here, data for an offset and/or rotation coordinate transformation or a general matrix operation can be inputted by the **McBSP interface**.

The following variants are provided for different use cases:

- The previous **"Local Online Positioning"** concerns the scan system-specific coordinate transformations in the real **Image field** analogous to **set_offset**, **set_angle** and **set_matrix** with parameter `HeadNo = 1` or `2` according to [5](#) and [6](#) in [Chapter 7.3.6 "Output Values to the Scan System", page 180](#). It is described in [Chapter 8.3.1 ""Local Online Positioning""](#), page [227](#).
- The **"Global Online Positioning"** concerns global coordinate transformations in the virtual **Image field** analogous to **set_offset**, **set_angle** and **set_matrix** each with `HeadNo = 4` according to [3](#) in [Chapter 7.3.6 "Output Values to the Scan System", page 180](#). It is described in [Chapter 8.3.2 ""Global Online Positioning""](#), page [230](#).

Online Positioning cannot be combined with Processing-on-the-fly applications with position information, but it can be combined with encoder-based Processing-on-the-fly applications.

Notes

- Since coordinate transformations [5](#) and [6](#) are calculated after the Processing-on-the-fly correction [4](#), larger **Image field** rotations are not well compatible with linear Processing-on-the-fly corrections. In such cases, **"Global Online Positioning"** is preferable.

8.3.1 "Local Online Positioning"

Reading in data for **"Local Online Positioning"** via the **McBSP interface** needs to be configured by **set_mcbsp_x**, **set_mcbsp_y** and/or **set_mcbsp_rot** or **set_mcbsp_matrix** (or by the equivalent list commands).

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 223](#) 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 [Chapter 9.3.2 "Conditional Command Execution", page 294](#)).

The **McBSP interface** is described in [Chapter 4.6.6 "McBSP/ANALOG Socket Connector", page 83](#).

Configuring "Local Online Positioning"

`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 interface** are interpreted by the RTC6 PCIe Board and which internal memory location is used to read the values:

- Depending on which of the above commands is called, the RTC6 PCIe Board interprets the inputted values as offsets in the x direction and/or y direction 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 4 options becomes activated, all values subsequently inputted at the **McBSP interface** are internally stored in memory location 1 or 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 or 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** 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 signed 16-bit 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** 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** input values are then alternatingly transferred to internal memory locations 1 and 2. The RTC6 PCIe Board 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 and 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** input values are likewise alternatingly transferred to internal memory locations 1 and 2.

The RTC6 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 signed 32-bit values. The indices are encoded in the data word (see command description). **McBSP** input values get transferred to internal memory location 1.

Notes

- You can use “**Local 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** signals:
 - When you use the commands for configuring “**Local 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** input values are copied to internal memory locations 1 and 2 (see above) and are then available for “**Local Online Positioning**”.
 - 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 “**Local Online Positioning**”. Subsequently, **McBSP** input values are then transferred to the internal memory locations 0 to 3 depending on the command and are then available for the Processing-on-the-fly application.
 - If you switch off (intentionally or with an invalid scaling factor) “**Local 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).

8.3.2 “Global Online Positioning”

“Global Online Positioning” (available as of RTC6 Software Package \geq V1.6.1) needs to be activated by one of the following commands:

- `set_mcbsp_global_matrix`
- `set_mcbsp_global_rot`
- `set_mcbsp_global_x`
- `set_mcbsp_global_y`
- `set_mcbsp_global_matrix_list`
- `set_mcbsp_global_rot_list`
- `set_mcbsp_global_x_list`
- `set_mcbsp_global_y_list`

Once activated, all other McBSP processings are deactivated (“Processing-on-the-fly” applications controlled by McBSP signals, “Local Online Positioning” as described in Chapter 8.3.1 ““Local Online Positioning””, page 227, processings activated by `set_mcbsp_in` or `set_multi_mcbsp_in`) and vice versa.

All subsequent data transferred via McBSP are internally handled in the same way as with “Local Online Positioning” (copied to internal memory locations 1 and possibly 2; readable by `read_mcbsp`),

but instead of the scan system-specific coordinate transformations in the real `Image` field (see Chapter 8.3.1 ““Local Online Positioning””, page 227) automatically used for coordinate transformations in the virtual `Image` field instead.

Calling `apply_mcbsp` or `apply_mcbsp_list` is no longer necessary and even has no effect.

Coordinate transformations in the virtual `Image` field (see also Chapter 8.6.4 “Compensating 2D Motions”, page 248) are automatically applied, as soon as a Processing-on-the-fly application is activated afterwards.

During a Processing-on-the-fly application, new data can only be sent and stored, but not applied, see also:

- `set_matrix(HeadNo = 4)`
- `set_offset_xyz(HeadNo = 4)`
- `set_angle(HeadNo = 4)`

“Global Online Positioning” is compatible with Processing-on-the-fly applications controlled by encoder signals.

Notice!

- The latest transferred data value is used immediately according to the current “Global Online Positioning” mode. Therefore, make sure to transmit correct values after changing that mode and before applying the data by starting a Processing-on-the-fly session.
- Example: `set_mcbsp_global_x` and `set_mcbsp_global_y` combine the xy offsets in the lower and upper half word of the transmitted data. `set_mcbsp_global_y(Scale = 0.0)` disables the y offset and the latest sent data value is used *in total* as x offset. Make sure to transmit the correct x offset again.

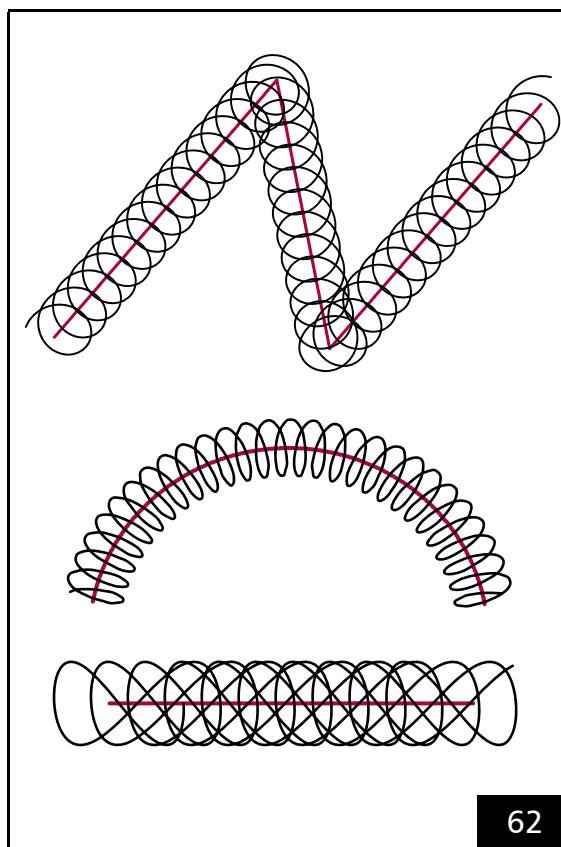
Configuring “Global Online Positioning”

The “Global Online Positioning” is configured the same way as the “Local Online Positioning”, see Chapter 8.3.1 ““Local Online Positioning””, page 227.

8.4 Wobble Mode

The **Wobble Mode** allows varying the line width for laser marking.

For this purpose, for example, 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 62](#). Alternatively, the motion can be combined with a horizontal or vertical figure-of-8.



62

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. If there is no direction of movement, no wobble movement is performed.

During Processing-on-the-fly correction by the [McBSP interface](#) or encoder interface where the scan head only compensates differences between actual external movements and intended total motion, see [Chapter 8.6.2 "Compensating Linear Movements", page 242](#), a slight jitter in the direction of galvanometer scanner 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. This is also particularly important if the complete translation movement takes place outside and the galvanometer scanners only have to carry out the actual sweep movement.

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](#) and after interruptions (for example, by 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^{\(1\)}](#)
- [laser_on_list](#)

For further details, see [set_wobble](#) and [set_wobble_mode](#).

(1) See also [set_wobble_mode_phase](#).

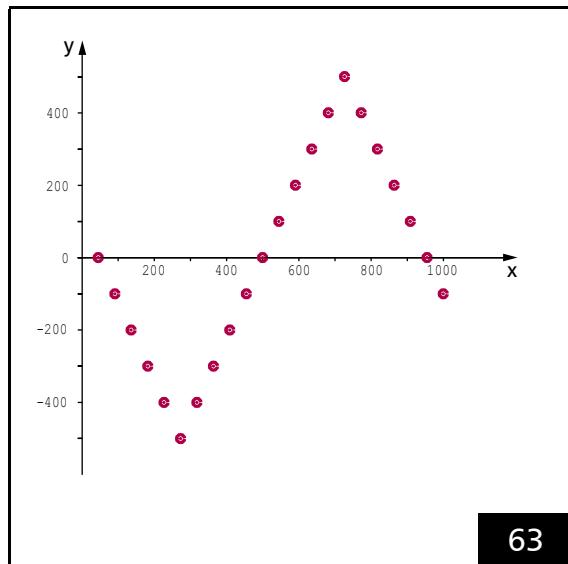
For optimum marking results, the wobble frequency and mark speed must complement each other, see [Chapter 8.4.1 "Wobble Shapes – Important Notes on Choosing Appropriate Parameter Values", page 233](#).

For many welding applications, the default set of "classic" wobble shapes (circle, ellipse, sine, figure-8) does not produce optimal results in the area of the weld seam. For example, high-speed motion occurs parallel to translation movements and low-speed motion occurs in the opposing direction.

set_wobble_vector lets you define a wobble shape customized for your user program, consisting of up to 1023 piecewise linear sections, while also specifying variation of laser power along that shape (see also **set_wobble_control**). By **create_dat_file**, this "freely definable wobble shape" is saved, see comment on [page 348](#).

However, **set_wobble_vector** cannot be combined with automatic or vector-based laser control.

The present wobble excursion from **set_wobble** or **set_wobble_mode** can be recorded by **set_trigger/set_trigger4** (signal 53). The format of the data is ((transversal << 16) + longitudinal).



63

See [Section "Example Code \(C++\)", page 232](#).

Example Code (C++)

The following example C++ source code shows a zigzag pattern, see [Figure 63](#).

The code must be included in a user program, see [Chapter 6.2.5 "Example Code \(C\)", page 99](#).

```
// Zigzag pattern

// Transversal micro step
const double dTrans( 100.0 );
// Longitudinal micro step
const double dLong( 0.0 );

// Number of steps
const uint16 Period( 5 );

// Start a new wobble shape
set_wobble_vector( 0, 0, 0, 0 );

// 1st wobble vector
set_wobble_vector( dTrans, dLong,
                    Period, 0.0 );
// 2nd wobble vector
set_wobble_vector( -dTrans, dLong,
                    Period * 2, 0.0 );
// 3rd wobble vector
set_wobble_vector( dTrans, dLong,
                    Period, 0.0 );
// if laser power variation is needed,
// include here
// set_wobble_control( Ctrl, Value,
//                      MinValue, MaxValue );

// Activate freely definable wobble shapes
set_wobble_mode( 100, 0, 100, 2 );

// Mark a vector
timed_mark_rel( 1000, 0, 22*10 );
```

8.4.1 Wobble Shapes – Important Notes on Choosing Appropriate Parameter Values

“Classic” Wobble Shapes

“Classic” wobble shapes are defined by `set_wobble_mode` or `set_wobble`.

Only assign hardware appropriate values to the `Transversal`, `Longitudinal` and `Freq` parameters.

Notice!

- Too big values define control situations where very high waste heat is produced. Galvanometer scanner 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 galvanometer scanners 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 $V_{Path} = 2 \times \pi \times A \times F$.

- Make sure that the combination of the used values and the trajectory⁽³⁾ 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⁽⁴⁾ you can also read-out the present temperatures of galvanometer scanners and/or digital control boards (see `control_command`).

Example of Use

System Specification

- **Tracking error** $T = 0.33$ ms.
- Calibration ± 0.349 rad_{optical}
($= 10^\circ$ mechanical) at $\pm 503,316$ bit.
- This is an angle control AC of
 $\approx 1.44 \times 10^6$ bit/rad_{optical}
($\approx 50,300$ bit/ $^\circ$ mechanical).
- Calibration factor⁽⁵⁾ $K = 5,000$ bit/mm.
- Typical positioning speed
 $V_{rad} = 100$ rad_{optical}/s.
- Converted to control values this corresponds to a typical positioning speed of
 $V = V_{rad} \times AC$
 $\approx 1.44 \times 10^8$ bit/s = $1,440$ bit/ $10\ \mu s$.
- In the **Image field** this corresponds to a typical positioning speed of $V/K = 28.8$ m/s.

(1) See technical specifications in the scan head manual.

(2) At the maximum frequency estimated with (1).

(3) See Glossary entry on [page 29](#).

(4) See Glossary entry on [page 26](#).

(5) See `*_ReadMe.txt` 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* mark 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 (!)

- should be lower than the specified maximum positioning speed,
- but there may be certain circumstances where it is much higher than the *typical* mark 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” (see **set_wobbel_vector**) take the dynamics of the scan head into account.

- The maximum repetition rate for “**Freely Definable Wobbel Shapes**” corresponds to the maximum wobbel frequency estimated by the rule of thumb 1, [page 233](#).
- The sum of the path (**Trajectory**) velocity and the velocity on the freely defined wobbel figure should not exceed the maximum positioning velocity.
- The acceleration should not significantly exceed the value of 2.5 V/S (where $V =$ typical positioning speed and $S =$ **Tracking error**).
- Also with “freely definable wobbel shapes”, observe the heating of the scan system for freely definable wobbel figures, see safety notice on [page 233](#) and rule of thumb 3.



8.5 Controlling 2D Scan Systems and 3D Scan Systems

8.5.1 2D Scan Systems

Up to two 2D scan systems can be controlled by one RTC6 PCIe Board.

The second one requires the [Option "Second Scan Head Control", page 37](#).

When controlling two 2D scan systems, a separate **Image field** correction can be carried out for each of the both scan systems.

By **select_cor_table** or **select_cor_table_list**, the two correction tables are assigned to the respective scan head connector, see also [Section "2. SCANHEAD Socket Connector", page 67](#).

To use two correction tables in a double scan system configuration

(1) Load each of the desired 2D correction files by

```
load_correction_file(Name, n, 2)  
(n = 1...8).
```

See also [Chapter 8.5.3 "Using Several Correction Tables", page 240](#).

(2) Assign a loaded 2D-correction table to the first and the second scan head each by calling

```
select_cor_table(HeadA, HeadB)  
with (HeadA and HeadB = 1...8).
```

(3) By **set_offset**, **set_scale**, **set_angle** or the corresponding list commands, specify offset, scaling factor and rotation to align the two **Image fields** precisely with respect to each other.

Notes

- If you are not using one of the scan head connectors: assign the correction table 0 to it.
- The default setting for **select_cor_table** and **select_cor_table_list** is (1, 0):
 - For the first scan head, correction table #1 is used
 - At the second scan head, there are *no position outputs*
- The RTC6 PCIe Board returns to this setting after every **load_program_file**. If a different setting is to be used, **select_cor_table** or **select_cor_table_list** must be called again.
- The scan head connectors *cannot* be simultaneously assigned:
 - two 3D correction tables
 - a 2D correction table *and* a 3D correction table

8.5.2 3D Scan Systems

An RTC6 PCIe Board can also be used to control a 3-axis scan system⁽¹⁾. This requires the [Option "3D", page 37](#).

Intended Use

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. Therefore, they are frequently used in applications for which flat field objectives are not available.

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 workpiece processing in 3 dimensions.

SCANLAB offers dynamic focusing units⁽²⁾ that extend *xy scan systems* to 3-axis scan systems.

Connection and Initialization

In order to control a 3-axis scan system by an RTC6 PCIe Board:

- The [Option "3D", page 37](#) of the RTC6 PCIe Board must be enabled (this can be checked by `get_rtc_version`)
- The *xy* scan system must be connected to the first scan head connector
- The *z* axis must be connected to the second scan head connector
- A 3D correction table (`D3_*.CT5`) must exclusively be assigned to the first scan head connector (by `select_cor_table` or `select_cor_table_list`).
- If, in addition, the [Option "Second Scan Head Control", page 37](#), is enabled, it is alternatively possible to connect:
 - the *xy* scan head to the second scan head connector
 - the *z* axis to the first scan head connector
 - In this constellation the 3D correction table must be assigned to the second scan head connector.

See also [Chapter 4.5 "Interfaces to Scan System", page 66](#) and [Section "2D and 3D Correction Files", page 173](#).

Other than that, no additional drivers or software files are needed for controlling a 3-axis scan system. Even initialization and program launching remain unchanged, see [Chapter 6.2 "Initialization and Program Start-Up", page 94](#).

Notice!

- The standard [RTC6 DLL \(RTC6DLL.dll\)](#) supports all RTC6 commands for controlling 3-axis scan systems. However, the full functionality of the RTC6 commands – the actual *output* of *z* coordinates – is only available with enabled [Option "3D", page 37](#).

Several 3-axis scan systems can be simultaneously controlled (by multi-board commands) from a single PC. This requires a corresponding number of RTC6 PCIe Boards with enabled [Option "3D"](#)s to be installed in that PC.

(1) Consisting of an *xy* scan system and a *z* axis dynamic focusing unit (see footnote 2) as *z* axis.

(2) See Glossary entry on [page 26](#).

3D Commands

These are, for example, the following RTC6 commands:

- `[*]3d[*]`
- `goto_xyz`
- `set_offset_xyz` and `set_offset_xyz_list`

Except for the additional motion in the third dimension, the 3D commands function identically to their corresponding 2D commands:

- Specified vectors and arcs are split-up into **Microsteps**, see [Chapter 7.1.2 "Microstepping", page 139](#)
- The jump speed and mark speed are specified by `set_jump_speed` and `set_mark_speed`, see [Chapter 7.1.1 "Marking with Vector Commands and "Arc" Commands", page 135](#). As for timed **Vector commands**, the speeds are automatically determined from 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 [Chapter 7.3.5 "Image field Correction and Correction Tables", page 172](#)
- For **Jump commands** and `[*]mark[*]` Commands, the laser control signals are switched on and off while taking delay settings into account, see [Chapter 7.2 "Delay Settings – Coordinating Scan Head Control and Laser Control", page 144](#)

For the value range of the data, see [Section "Compatibility Modes", page 167](#).

The calibration factor K_{xy} can be read out from the correction table used by `get_table_para`.

In [RTC4 Compatibility Mode](#) and [RTC5 Compatibility Mode](#), the three coordinate values are always scaled up to 20-bit values internally, so that the three spatial directions are treated equally with regard to the jump speed or mark speed.

The 3 coordinate values are internally always scaled up to 20-bit values, so that the three spatial directions are treated equally with regard to the jump speed or mark speed.

With big z jumps, observe the different dynamics of varioSCAN and 2D scan systems.

Notes

- After "vector-controlled laser control" has been activated by `set_vector_control` (`Ctrl = 7`), the para-**Mark command** and para-**Jump commands** can be used to set a dynamic focus shift linearly along the mark or jump vector, see [Section "Vector-Defined Laser Control", page 205](#). See also `set_defocus` or `set_defocus_list`.
- The size of the usable **Image field**⁽¹⁾ and the focus shift in the z direction⁽²⁾ (height of the usable **3D image field**) can be obtained from the `*_ReadMe.txt` file supplied with the 3D correction file as well as from the user manual of the 3-axis scan system/varioSCAN ("Technical Specifications" chapter).
- If a z axis is to be used to hold the laser focus only in a certain plane (especially in 3D systems without a lens), then even 2D vector commands can be used. 2D vector commands leave the z position previously set with a 3D vector command unchanged and only adjust the focus length.
- If the [Option "3D", page 37](#) is not enabled or no 3D correction table has been assigned:
 - The split-up into **Microsteps** is calculated including the z axis (which influences the effective jump speed and mark speed in the xy plane).
 - There is merely no output for the z axis

(1) Line "Max. Field Size (z=0): nnn.nnn mm".

(2) Line "Max. Z-Range: +/- nn.n mm".

- By `set_offset_xyz` or `set_offset_xyz_list`, an offset can be defined 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 the offset).
- During a marking process, the scan system focal length is continuously readjusted. During execution of a `Jump command` (or `goto_xyz`) this readjustment can be switched off. With `DirectMove3D = 1` in `set_delay_mode` or `set_delay_mode_list`, the z output value is directly (linearly) taken to its final value during the jump.

Adjusting Tracking Errors

If the xy scan head and the **Dynamic focusing unit** unit of a 3D scan system have different **Tracking errors**, `set_timelag_compensation` can be used to *delay* the output of the faster component to that of the slower component.

The laser control signal output is synchronized with the slower component. This allows, for example, **Sky Writing** with 3D commands without permanent defocusing.

In case of a **SCANahead system**, the `PreviewTime` parameter from `set_scanahead_params` is automatically used as "**Tracking error**".

Enhanced 3D Correction

- For more information on the actual procedure, refer to the "Calibrating a 3-Axis Laser Scan System" Manual, Chapter 3.4 "Step 4: Stretch factor correction".

The image size of 3D scan systems without objectives or with non-telecentric F-Theta objectives depends on its distance to the scan head objective (z coordinate).

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 **Section "ct5 Correction File Header", page 177**.

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 (2D stretch correction table). 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:

$$\begin{aligned}<\text{StretchX}> &= (X-X')/Z \\ <\text{StretchY}> &= (Y-Y')/Z\end{aligned}$$

You can then use **load_stretch_table** to load the enhanced 3D correction onto the RTC6 PCIe Board from an ASCII text file and assign it to an already loaded 3D-correction table. This ASCII text 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.

Enhanced 3D correction is not active by default after **load_program_file**. It becomes active upon loading a valid table onto the RTC6 PCIe Board. It can be deactivated by calling **load_stretch_table** using a **NULL** pointer instead of the filename.

For the 2D stretch correction tables, the following rules apply:

- The ASCII text file can contain one or several tables.⁽¹⁾
- The 2D stretch correction table must begin with the line:
`[StretchTable<No>]`
`<No>` represents the table number to be specified by **load_stretch_table**.
 - 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.
 - Empty lines or incomplete data lines are invalid and are ignored.

(1) Even of another type, see table 1, page 152.



8.5.3 Using Several Correction Tables

The RTC6 PCIe Board memory can store 8 different correction tables at the same time.

It can be useful to work with several different correction tables even if only a single scan system (2D or 3D) is used. Example: a pilot laser and a working laser with different wavelengths are used.

Special applications sometimes also require different correction tables in quick succession. This change can be done, for example, by `select_cor_table(n, 0)` or `select_cor_table_list(n, 0)` ($n = 1 \dots \text{number_of_correction_tables}$).

`number_of_correction_tables` serves to protect other commands (for example, `load_correction_file` and `select_cor_table`) from unwanted table numbers.

There is no need to change existing user programs except when user input is to be rejected in the future (by means of explicit RTC6 error messages).

8.6 Processing-on-the-fly

- “Processing-on-the-fly” means the combination of workpiece movement and scan system movement.
- The use of the *Processing-on-the-fly* functionality requires the [Option Processing-on-the-fly](#).
- Whether the [Option Processing-on-the-fly](#) is enabled can be checked by [get_RTC_version](#).
- [Chapter 8.6.12 “Fly Extension” Commands](#), page 258
- Available are (not mixable):
 - “Classic” Processing-on-the-fly commands⁽¹⁾
 - “Fly Extension” Commands, page 258⁽²⁾

8.6.1 Intended Use and Initialization

With its [Option Processing-on-the-fly](#) enabled, the RTC6 PCIe Board allows processing of parts in motion (for example, parts on a conveyor belt, rotating plate or xy positioning stage), as well as stationary parts with a moving scan system (for example, by a robot arm).

To adjust laser scan processes to the current workpiece position relative to the scan system, the position of the workpiece or scan system can be forwarded to the RTC6 PCIe Board:

- indirectly by encoder counters
- directly by the [McBSP interface](#)

If the Processing-on-the-fly correction is active, the coordinate values of all [Vector commands](#) and [“Arc” commands](#) are transformed based on the forwarded position values.

Upon forwarding the movement as encoder pulses, RTC6-internal encoder counters are triggered. The counter values correspond to the current position. They get a scaling factor (from certain RTC6 commands) assigned and are then used as *Processing-on-the-fly* correction.

See also [Chapter 9.3.3 “Synchronization by Encoder Signals](#), page 297.

Upon forwarding the position values via the [McBSP interface](#), the input values get a scaling factor assigned and are then used as *Processing-on-the-fly* correction.

See also [Chapter 9.3.4 “Synchronization and Online Positioning by McBSP Signals](#), page 299.

Simultaneous usage of both forwarding methods for Processing-on-the-fly correction of two independent motions is not possible, see [Section “Overview”](#), page 241.

The [McBSP interface](#) cannot be simultaneously used for a Processing-on-the-fly application and an [Online Positioning](#), see [Notes](#), page 229.

Processing-on-the-fly correction is activated and deactivated by list commands. The parameters required for activation may have to be determined beforehand by a calibration process (see below).

If both scan head connectors each have a 2D correction table assigned, then a Processing-on-the-fly correction has the same effect at both scan head connectors.

For the z axis, a Processing-on-the-fly correction can be activated as well, see [Chapter 8.6.11 “Processing-on-the-fly Correction for the z Axis](#), page 256.

Overview

The following encoder-based Processing-on-the-fly corrections of motions are available:

- **set_fly_x, set_fly_y**, even in combination
To compensate linear motions of the workpiece (for example, by conveyor belt or xy positioning stage), see [Chapter 8.6.2 “Compensating Linear Movements](#), page 242.
- **set_fly_rot**
To compensate rotary motions of the workpiece (for example, by rotating plate), see [Chapter 8.6.3 “Compensating Rotary Movements](#), page 246.
- **set_fly_2d**
To compensate 2D motions of the workpiece (for example, xy positioning stage), see [Chapter 8.6.4 “Compensating 2D Motions](#), page 248.

(1) See [“Classic” Processing-on-the-fly Control Commands](#), page 307 and [“Classic” Processing-on-the-fly List Commands](#), page 311.

(2) See [“Fly Extension” List Commands](#), page 312.

The following **McBSP**-based Processing-on-the-fly corrections of motions are available:

- **set_fly_x_pos, set_fly_y_pos** even in combination
To compensate 1D or 2D motions of the scan system (for example, robot arms), see [Chapter 8.6.2 "Compensating Linear Movements", page 242](#).
- **set_fly_rot_pos**
To compensate rotary motions of the scan system (for example, rotary table), see [Chapter 8.6.3 "Compensating Rotary Movements", page 246](#).
- **set_mcbsp_in, set_mcbsp_in_list**
To compensate 1D or 2D motions or rotary motions (for example, robot arms, rotary table), see [Chapter 8.6.2 "Compensating Linear Movements", page 242](#) and [Chapter 8.6.3 "Compensating Rotary Movements", page 246](#).

The following Processing-on-the-fly corrections can be combined:

- **set_fly_x** with **set_fly_y**.
- **set_fly_x_pos** with **set_fly_y_pos**
- For the special case **set_fly_z**, see [Chapter 8.6.11 "Processing-on-the-fly Correction for the z Axis", page 256](#)
- For linear 3D Processing-on-the-fly corrections with positional values, see [Section "Correction via McBSP Interface with Enhanced McBSP Input", page 245](#).

The following Processing-on-the-fly corrections *cannot* be combined:

- Encoder-based with **McBSP**-based
- linear and rotating

The last command always determines the overall correction unless it can be combined with previous settings.

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 8.6.7 "Synchronizing Processing-on-the-fly Applications", page 251](#).

8.6.2 Compensating Linear Movements

Processing-on-the-fly correction for linear movements (translations) can be activated⁽¹⁾ by:

- **set_fly_x** and/or **set_fly_y**
- **set_fly_x_pos** and/or **set_fly_y_pos**
- **set_mcbsp_in** (Mode = 1...3)
- **set_mcbsp_in_list** (Mode = 1...3)

A scaling factor must thereby be specified in each case.

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 deactivated (simultaneously for both directions) by **fly_return**.

See the corresponding notes in [Chapter 8.6.5 "Deactivating Processing-on-the-fly Correction", page 250](#).

Correction via Encoder Counters

To be able to pass position values via the board-internal encoder counters⁽²⁾, the 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 also [Section "Determining the Scaling Factors", page 243](#).

By **set_fly_x**, the encoder counter "Encoder0" is reset to zero. By **set_fly_y** the encoder counter "Encoder1" is reset to zero.⁽³⁾

(1) Different Processing-on-the-fly corrections cannot be combined arbitrarily, see the [Section "Overview", page 241](#).

(2) See [Notice!](#), page 56.

(3) By **set_control_mode** Bit #9 it can be set beforehand whether the counter is reset at the respective Processing-on-the-fly command or at the start trigger.

Thereafter, the output value is calculated from the current output position by adding (for all spatial directions) the product of the scaling factor and the current counter value.

This correction takes place every 10 µs.⁽¹⁾

Notes

- `set_fly_x` and `set_fly_y` can be used in combination for 1D as well as for 2D Processing-on-the-fly applications (for example, an xy positioning stage), particularly for separate or separable marking tasks in the real **Image field**.
- For continuous marking in the virtual **Image field**, SCANLAB recommends to preferably use `set_fly_2d`, see [Chapter 8.6.4 "Compensating 2D Motions", page 248](#).

Determining the Scaling Factors

The scaling factors can be determined experimentally:

- (1) Read out the counter start value by `get_encoder`.
- (2) Start the movement.
- (3) Stop the movement.
- (4) Read out the counter end value by `get_encoder`⁽²⁾.
- (5) Measure the distance travelled in mm.
- (6) Calculate the encoder increment i as follows:

$$i = \frac{(\text{counter end value} - \text{counter start value})}{\text{distance travelled}}$$

- (7) Calculate the scaling factors Scale_x and Scale_y [in bits per count] as follows:

$$\text{Scale}_x = \frac{K}{i_x}$$

$$\text{Scale}_y = \frac{K}{i_y}$$

Whereby K is the calibration factor [in bits per mm], see [Chapter 7.3.2 "Image Field Size and Image Field Calibration", page 166](#) and i is the encoder increment from Step 6.

If the workpiece moves at a constant speed v_x or v_y [in mm per second] and an encoder simulation has been activated by `simulate_encoder`, then the scaling factors are calculated as follows:

$$\text{Scale}_x = \frac{K \times v_x}{(1.000.000 \text{ counts} / \text{s})}$$

$$\text{Scale}_y = \frac{K \times v_y}{(1.000.000 \text{ counts} / \text{s})}$$

- (8) Adjust the signs of the scaling factors according to the movement direction.

(1) The encoder counters are signed 32-bit counters with overflow (= after reaching the maximum (minimum) counter value, counting continues at the minimum (maximum) counter value).

(2) Alternatively, the counter start and end values can be stored in a cache on the RTC6 PCIe Board by the list command `store_encoder` and then retrieved from there by the control command `read_encoder`.

Correction via McBSP Interface

To be able to pass position values via the **McBSP interface**, the Processing-on-the-fly correction must be activated by:

- **set_fly_x_pos** and/or **set_fly_y_pos**
- **set_mcbsp_in** for positional values in 2 spatial directions
- **set_multi_mcbsp_in** for positional values in all 3 spatial directions (requires no scaling factors)

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 interface**. See also [Section "Determining the Scaling Factors", page 244](#).

Thereafter, the output value is calculated from the current output position by adding (for all spatial directions) the product of the scaling factor and the current input value at the **McBSP interface**.

This correction takes place every 10 µs.

At the **McBSP interface**, see [Chapter 4.6.6 "McBSP/ANALOG Socket Connector", page 83](#), there are available:

- For 1D corrections – signed 32-bit values
- For 2D corrections – signed 16-bit values per axis (the y value is in the lower 16 bits and the y value in the upper 16 bits of the value at the **McBSP interface**)

Notes

- After **set_fly_x_pos** and **set_fly_y_pos**, the input values received at the **McBSP 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 **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** also result in **McBSP** input values being copied to internal memory locations 1, 2 and/or 3, see [Section "Correction via McBSP Interface with Additional McBSP Input", page 245](#).
- 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...3. From there, they can be read out unsorted by **read_mcbsp** or sorted by **read_multi_mcbsp**.

Determining the Scaling Factors

The scaling factors can be determined experimentally:

- (1) Read out the start input value by **read_mcbsp**.
- (2) Start the movement.
- (3) Stop the movement.
- (4) Read out the end input value by **read_mcbsp**.
- (5) Measure the distance travelled in mm.
- (6) Calculate the position increment i as follows:

$$i = \frac{(\text{end input value} - \text{start input value})}{\text{distance travelled}}$$

- (7) Calculate the scaling factors **Scalex** and **Scaley** [in bits per bits] as follows:

$$\text{Scalex} = \frac{K}{i_x}$$

$$\text{Scaley} = \frac{K}{i_y}$$

Whereby K is the calibration factor [in bits per mm], see [Chapter 7.3.2 "Image Field Size and Image Field Calibration", page 166](#) and i is the encoder increment from Step 6.

Correction via McBSP Interface with Additional McBSP Input

To activate Processing-on-the-fly correction for linear motions using **McBSP** 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 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** 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`.
- A scaling factor is specified at `set_mcbsp_in` or `set_mcbsp_in_list`. For 2D corrections (`Mode = 3`), 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 1D 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 2D 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 interface**. For a description of the interface, see **Chapter 4.6.6 "McBSP/ANALOG Socket Connector"**, page 83.

Correction via McBSP Interface with Enhanced McBSP 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 direction and y direction, but also in the z direction and with laser power variation. Furthermore, up to 4 additional signals can be transmitted for usage as you wish.

Each 10 μ s, data asynchronously transmitted to **McBSP** 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 sorted by signal types using `read_multi_mcbsp`.

8.6.3 Compensating 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:

- `set_fly_rot`
- `set_fly_rot_pos`
- `set_mcbsp_in` (Mode = 4)
- `set_mcbsp_in_list` (Mode = 4)

Processing-on-the-fly correction can be stopped by `fly_return` (see the corresponding notes on Chapter 8.6.5 "Deactivating Processing-on-the-fly Correction", page 250).

Notes

- A Processing-on-the-fly rotation correction:
 - Cannot be combined with other Processing-on-the-fly corrections, see [Section "Overview", page 241](#).
 - Simultaneously for both scan head connectors is only practical, if both attached scan systems are aligned to exactly the same rotation center.

Correction via Encoder Counter

To be able to pass rotation angles via the board-internal incremental encoder, the encoder input `ENCODER X`⁽¹⁾ must be connected, see also [Section "Input Ports for External Encoder Signals", page 298](#).

Then, Processing-on-the-fly correction must be activated by `set_fly_rot`. The parameter `Resolution` [in counts per revolution] must thereby be specified. See also [Section "Determining the Resolution Parameter", page 246](#).

`set_fly_rot` resets the encoder counter "Encoder0" to zero.⁽²⁾

(1) `ENCODER X` = Encoder0.

(2) By `set_control_mode` Bit #9 it can be set beforehand whether the counter is reset at the respective Processing-on-the-fly command or at the start trigger.

The output value is calculated from the current output position via a rotation matrix around the specified rotation center with rotation angle / 360° = current "Encoder0" counter reading / Resolution).

This correction is performed every $10 \mu\text{s}$.⁽³⁾

Determining the Resolution Parameter

The `Resolution` parameter can be determined experimentally:

- (1) Read out the counter start value by `get_encoder`.
- (2) Start the rotation.
- (3) Count the number of revolutions.
- (4) Stop the rotation.
- (5) Read out the counter end value by `get_encoder`.⁽⁴⁾
- (6) Calculate the parameter `Resolution` [in counts per revolution] as follows:

$$\text{Resolution} = \frac{(\text{counter end value} - \text{counter start value})}{\text{number of revolutions}}$$

$$\text{Resolution} = (\text{counter end value} - \text{counter start value}) / \text{number of revolutions}$$

If the workpiece rotates at a constant speed ω [in number of revolutions per second] and an encoder simulation has been activated by `simulate_encoder`, then the `Resolution` parameter can be calculated as follows:

$$\text{Resolution} = \frac{(1.000.000 \text{ counts} / \text{s})}{\omega}$$

- (7) Adjust the sign of `Resolution` to the direction of rotary movement.

(3) The encoder counters are signed 32-bit counters with overflow (= after reaching the maximum (minimum) counter value, counting continues at the minimum (maximum) counter value).

(4) Alternatively, the counter start and end values can be stored in a cache on the RTC6 PCIe Board by the list command `store_encoder` and then retrieved from there by the control command `read_encoder`.

Correction via McBSP Interface

If angle-position values for Processing-on-the-fly correction of rotary movement are forwarded by the **McBSP interface**, then Processing-on-the-fly correction must be activated by `set_fly_rot_pos`.

The required `Resolution` parameter has the same meaning as with `set_fly_rot` and is similarly determined, see [Section "Determining the Resolution Parameter", page 246](#).

McBSP input values are read out by `read_mcbsp`.

Notes

- McBSP input values get copied to board-internal memory location 0, see notes in [Section "Correction via McBSP Interface", page 244](#).

Correction via McBSP Interface with Additional McBSP Input

A Processing-on-the-fly correction for rotary movements with McBSP input values can be activated by:

- `set_mcbsp_in` (Mode = 4)
- `set_mcbsp_in_list` (Mode = 4)

These commands offer the advantage of using the **McBSP 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 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`.
- By `set_mcbsp_in` or `set_mcbsp_in_list`, a rotation resolution can be specified. Calibration for determining the rotation resolution is the same as for `set_fly_rot_pos`, see [Section "Determining the Resolution Parameter", page 246](#).
- For transferring rotary correction values, 31 bits with sign are effectively available.

8.6.4 Compensating 2D Motions

You can use the `set_fly_2d` command to activate encoder-based 2D Processing-on-the-fly correction (for example, for a xy positioning stage), particularly for continuous marking in the virtual `Image` field.

Compared to an activation by `set_fly_x` and `set_fly_y`, this has 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 positioning stage motions, see [Section "2D Encoder Compensation for xy Positioning Stages", page 248](#)
- 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 positioning stage encoder values (whereby the laser focus remains stationary relative to the xy positioning stage). Thus, unnecessary jumps are avoided after xy positioning stage forwarding motions, see [Chapter 8.6.6 "Virtual Image Field with Processing-on-the-fly", page 251](#)

Notes

- A `set_fly_2d` correction cannot be combined with other Processing-on-the-fly corrections, see [Section "Overview", page 241](#).
- Coordinate transformations within the virtual `Image` field, see [Section "Coordinate Transformations in the Virtual Image Field", page 168](#) can be activated when starting a `Processing-on-the-fly` session with changed values, if they have been loaded before with the corresponding control commands.

2D Encoder Compensation for xy Positioning Stages

For particularly demanding marking requirements, it might be necessary to also compensate mechanical deviations of a xy positioning stage.

For this purpose, `load_fly_2d_table` can be used to load up to two 2D compensation tables, see [page 249](#), onto the RTC6 PCIe Board.

Then – during a Processing-on-the-fly application initiated by `set_fly_2d` – encoder values are 2D interpolated and compensated just like with field correction tables.

To avoid unnecessary initialization motions, you can use `init_fly_2d` to define a desired positioning stage start position as the reference value for 2D encoder compensation (and to store it on the RTC6 PCIe Board) and to select simultaneously one of the both correction tables.

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 absolute position of the positioning stage is retained for compensation. This relation remains even upon termination with `fly_return` or upon a new start with `set_fly_2d`.

This relation does not remain, 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 original uncompensated encoder values use:

- `get_encoder`
- `store_encoder`
- `read_encoder`
- `wait_for_encoder`
- `wait_for_encoder_mode`
- `wait_for_encoder_in_range`
- Recording by `set_trigger` (`Signal1/Signal2 = 43 or 44`)
- Recording by `get_value`

For `load_fly_2d_table`, you need to provide an ASCII text file that contains a 2D compensation table.

A 2D compensation 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 positioning stage required by your application. Reference points with values exceeding $\pm 524,288$ 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 2D compensation table from an ASCII text file gets loaded onto the RTC6 PCIe Board. You can subsequently deactivate it by calling `load_fly_2d_table` using a `NULL` pointer instead of the filename.

For the 2D compensation tables, the following rules apply:

- The ASCII text file can contain one or several tables.⁽¹⁾
- The 2D compensation table must begin with the line:
`[Fly2DTable<No>]`
`<No>` stands for the table number `No`, which must be specified with `load_fly_2d_table`. The first of the two tables is loaded with `No = <No>` and the second with `No = <No> + 65,536`.
- 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 ASCII text file 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.

(1) Even of another type, see table 1, page 152.

8.6.5 Deactivating Processing-on-the-fly Correction

All Processing-on-the-fly corrections can be deactivated (simultaneously for both spatial directions) by **fly_return**.

fly_return requires a new output position to be specified, which then is reached by a normal jump.

Processing-on-the-fly correction that has been activated by **set_multi_mcbsp_in** should be terminated by **fly_return_z**.

In all other cases (see below) a “Hard jump” to a new output position may occur.

Notes

- If Processing-on-the-fly correction is not explicitly deactivated⁽¹⁾, then:
 - it is also effective during execution of subsequent lists
 - it is not effective in the pause between two lists⁽²⁾
- 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 with invalid parameter values. This could lead to a jump to an uncorrected output position (also refer to the command descriptions).
- If Processing-on-the-fly has been 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 241](#). This could lead to a “Hard jump” to a new output position.

- Processing-on-the-fly correction activated by **set_mcbsp_in** or **set_mcbsp_in_list** also gets deactivated if you call **set_mcbsp_in** with **Mode = 0** or **set_mcbsp_in_list** with **Mode = 0**. This could lead to a “Hard jump” to an uncorrected output position
- 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 [Section “Notes”, page 229](#). This could lead to a “Hard jump” to a new output position.
- A deactivation of a Processing-on-the-fly correction occurs only after the scanner delay.
- If Processing-on-the-fly has been activated by **set_mcbsp_in** or **set_mcbsp_in_list**, then the **fly_return** command deactivates Processing-on-the-fly correction, but it does not deactivate copying to internal memory locations (just like with **set_mcbsp_in**⁽⁵⁾). To afterward also deactivate copying to internal memory locations, you can use **set_mcbsp_in(0)** or **set_mcbsp_in_list(0)**.
- Processing-on-the-fly correction also gets deactivated (for both spacial directions simultaneously) by **stop_execution** or an [External Stop](#).

(1) **set_end_of_list** does not deactivate Processing-on-the-fly correction.

(2) Therefore, the correction does not affect the control commands **goto_xy** and **goto_xyz**.

8.6.6 Virtual Image Field with Processing-on-the-fly

With Processing-on-the-fly applications, the full value range (29-bit) of the virtual **Image field** can be used, see [Chapter 7.3.3 "Virtual Image Field", page 168](#) to load and process command lists with objects that are up to 512 times larger than the real 20-bit **Image field**.

With 1D Processing-on-the-fly applications (for example, workpieces on a conveyor belt), objects can only exceed the real **Image field** in the very dimension parallel to the Processing-on-the-fly direction.

With 2D Processing-on-the-fly applications (for example, with an *xy* positioning stage), the to-be-marked objects of a command list may be distributed across the entire virtual **Image field**.

If an entire marking task consists of several partial markings ("tiles") whose extent does not exceed the real **Image field**, the Processing-on-the-fly functionality can also only be used to move the partial marking to be marked into the real **Image field** and then process it there.

Coordinate values which are still outside the real image *after* Processing-on-the-fly correction are clipped to the boundary values of the real **Image field**. Here, the **get_marking_info** error bits get set automatically, see [Chapter 8.6.9 "Monitoring Processing-on-the-fly Corrections", page 254](#).

If there is a risk of the real 20-bit **Image field** being exceeded during list execution, specifically a forwarding motion of the conveyor belt or *xy* positioning stage should be executed. The forwarding motion can be waited for with **wait_for_encoder**, **wait_for_encoder_mode**, **wait_for_encoder_in_range** or **wait_for_mcbsp** by interrupting the list execution until certain encoder values or **McBSP** values are reached, see [Chapter 8.6.7 "Synchronizing Processing-on-the-fly Applications", page 251](#).

The appropriate break up of the entire marking task and synchronization with the 1D or 2D Processing-on-the-fly movement is the task of the user program.

8.6.7 Synchronizing Processing-on-the-fly Applications

Processing of command lists can be started by either a RTC6 command or an external start signal, see [Chapter 6.4.4 "Starting and Stopping Lists", page 108](#).

If the command list is not to be started immediately with the start signal, then an appropriate delay may be implemented as follows:

- When transferring positions via the **McBSP interface** – by a **wait_for_mcbsp** at the beginning of the command list.
 - When transferring positions via encoder counters – by a **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** at the beginning of the command list.
- wait_for_encoder_in_range** is useful for 2D encoder-based Processing-on-the-fly applications. **wait_for_encoder_in_range** waits until both encoders are within the specified range at the same time and thus does not depend on the actual **Trajectory** that used to reach this range⁽¹⁾

When transferring positions via encoder counters, a track delay can be configured (even independently of an active **Processing-on-the-fly session**) to delay the execution of the actual start relative to the triggering start signal or RTC6 command, see [Section "External Start", page 289](#).

(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 fulfillment of both encoder criteria would depend on the *xy* positioning stage motion's explicit **Trajectory** and you would need to define and reproduce an appropriate **Trajectory** even before loading the lists.

External 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 [Section "External Start", page 289](#)). 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 execute several lists at defined intervals.

If `wait_for_encoder`, `wait_for_encoder_mode`, `wait_for_encoder_in_range` and `wait_for_mcbsp` are used to interrupt a list for intermediate forwarding motions, see [Chapter 8.6.6 "Virtual Image Field with Processing-on-the-fly", page 251](#), then the [Signals for "Laser Active" Operation](#) are not changed before the forwarding motion.

With encoder-based Processing-on-the-fly applications, the laser focus with `park_position` can be moved to a safe parking position before an interruption and back to the starting position or any other position after an interruption with `park_return`. See command description for more details.

The behavior of the galvanometer scanners during such a forwarding motion depends on the Processing-on-the-fly correction used:

- With McBSP-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 relative to the real **Image field** during list interruption. As a rule, a jump to the next marking must then be made.
- With the encoder-based `set_fly_2d` Processing-on-the-fly correction, the positions of the galvanometer scanner are continuously Processing-on-the-fly-corrected (the laser focus thus remains stationary relative to the to-be-marked object). The subsequent jump to the next marking or continuation of the same is usually very small.
- With `set_fly_2d`, clipping might occur if the Processing-on-the-fly-corrected coordinate values exceed the real **Image field** during a long forwarding motion. To avoid this, it might make sense to switch off Processing-on-the-fly correction before the forwarding motion (by `fly_return`, see [Chapter 8.6.5 "Deactivating Processing-on-the-fly Correction", page 250](#); the galvanometer scanners then remain stationary). To switch correction back on after the forwarding motion, you can use `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 convert the last Processing-on-the-fly-uncorrected coordinate values such that the Processing-on-the-fly-corrected output matches the current output. If an error occurs when restarting with these commands (for example, because the recalculated coordinate values would then be outside the 29-bit virtual **Image field**, or because another Processing-on-the-fly mode has been activated in the meantime), then an error bit gets set. This error bit can be read out by `get_marking_info`(Bit #9).

If, during list processing, it is necessary to immediately respond to this error, then the list command **if_not_activated** can be called to possibly jump to an error-handling routine.

- **activate_fly_2d_encoder** or **activate_fly_xy_encoder** can be used to continue at another positioning stage position when switching on again. They reset the encoders and simulate a positioning stage position using the encoder values passed as parameters. This avoids larger forwarding motions for initialization.

8.6.8 Encoder Resets

Many encoder-based Processing-on-the-fly applications (for example, a conveyor belt continuously traveling in the same direction) need to occasionally reset the encoders (for example, before a new marking sequence). 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 encoder-based Processing-on-the-fly applications with continuous marking (for example, using an xy positioning stage in the virtual **Image field**), however, the relationship between the encoder values and the absolute position of the xy positioning stage should usually be preserved.

For this purpose, **set_fly_2d** should be used for Processing-on-the-fly activation. Before a long interruption, you can deactivate Processing-on-the-fly correction with **fly_return**. And after the interruption you can use **activate_fly_2d** or **activate_fly_2d_encoder** to resume correction. See also [Chapter 8.6.4 "Compensating 2D Motions", page 248](#) and [Chapter 8.6.7 "Synchronizing Processing-on-the-fly Applications", page 251](#). Processing-on-the-fly correction can also be resumed with **activate_fly_xy** or **activate_fly_xy_encoder**, but some functions are no longer available, particularly those necessary for the relation between current encoder values and absolute xy positioning stage positions, see the [Section "2D Encoder Compensation for xy Positioning Stages", page 248](#).

By **set_control_mode** Bit #9 it can be set beforehand whether the counter is reset at the respective Processing-on-the-fly command or at the start trigger.

8.6.9 Monitoring Processing-on-the-fly Corrections

The **Image field** boundaries might be reached, if:

- A Processing-on-the-fly user program is not optimized for the motion of the workpiece or scan system
- Considerable unintended change of workpiece or scan system speed occurs during a Processing-on-the-fly operation

Because the RTC6 PCIe Board 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 RTC6 PCIe Board sets internal error bits (**Bit #0...Bit #3**) if the **Image field** boundaries are exceeded (and coordinate values get clipped). These can be read out 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

- The error bits **Bit #0...Bit #3** can be used to determine which edge of the **Image field** has been exceeded. Each boundary exceedance results in setting of the corresponding error bit.
- The error bits **Bit #0...Bit #3** are reset during initialization by **load_program_file** and by **get_marking_info**. Therefore, **get_marking_info** returns information about errors that occurred since the last initialization or the last call of **get_marking_info**. Subsequent transformations of type 5...9, see [Chapter 7.3.6 "Output Values to the Scan System", page 180](#), are not taken into account here.
- During the adjustment phase of a Processing-on-the-fly correction, coordinate points at the edge of the **Image field** should therefore be approached and checked for possible limitations.
- The error bit **Bit #9** indicates if the 29-bit virtual **Image field** range has been exceeded during resumption of Processing-on-the-fly correction by **activate_fly_2d** or **activate_fly_xy**, see [Chapter 8.6.7 "Synchronizing Processing-on-the-fly Applications", page 251](#).



Customer-Defined Monitoring Area

For Processing-on-the-fly applications, the RTC6 PCIe Board also checks for exceedance of a second value range. Its boundaries can be specified by [set_fly_limits](#).

Such exceedances likewise result in setting internal error bits ([Bit #4...Bit #7](#)). These can be read out by [get_marking_info](#).

Moreover, the following conditional commands allow execution of any list command to be made dependent upon whether boundary exceedance in a customer-defined monitoring area occurred or not:

- [if_fly_x_overflow](#)
- [if_fly_y_overflow](#)
- [if_not_fly_x_overflow](#)
- [if_not_fly_y_overflow](#)

If the condition specified in the command parameter to the error bits [Bit #4...Bit #7](#) is fulfilled, the immediately following list command is executed. Otherwise, it is skipped.

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 [Bit #0...Bit #3](#)) only occurs if the maximum [Image field](#) ($-524,288\dots+524,287$ bit) is exceeded (the customer-defined monitoring area is typically smaller).
- The error bits [Bit #4...Bit #7](#) are reset during initialization (by [load_program_file](#)), but *not* by [get_marking_info](#). Individual error bits get implicitly reset by the conditional commands and can also be explicitly reset by [clear_fly_overflow](#) or [clear_fly_overflow_ctrl](#) (see command description).
- The error bits [Bit #4...Bit #7](#) do *not* take into account transformations of type [5...9](#), see [Chapter 7.3.6 "Output Values to the Scan System", page 180](#).
- Also for Rotation-fly applications it is possible to monitor a rectangular area, but not a rotation angle range.

8.6.10 Tracking Error Compensation of Encoder Values for Processing-on-the-fly Applications – NOT YET IMPLEMENTED

Not yet implemented.

8.6.11 Processing-on-the-fly Correction for the z Axis

The encoder-based Processing-on-the-fly correction for the z axis (referred to as “FlyZ correction” in the following) can be activated by `set_fly_z`.

You can add FlyZ to Processing-on-the-fly correction for the x axis and y axis that had been activated by `set_fly_x/set_fly_y/set_fly_rot` or can be effective on its own.

This makes dynamic marking possible if the workpiece plane ($z = 0$) and the scan system do not move parallel to each other.

With “FlyZ correction” activated, the z component workpiece motion gets compensated by re-adjustment of the z axis (see [Dynamic focusing unit](#)) in accordance with the current encoder signals.

Prerequisites

- Both, [Option Processing-on-the-fly](#) as well as the [Option “3D”](#) 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 commands.
 - The varioSCAN must be capable of tracking the xy galvanometer scanner motions of the scan system.
- For moving workpieces, the [Dynamic focusing unit](#) must also be capable of following workpiece motion. Take care to exceed neither the maximum focus range in z direction nor the depth of field of the objective. Ensure that the precision of the correction table 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 the z coordinate does *not* exist.
- See also the operational prerequisites for 3-axis scan systems in [Section “Connection and Initialization”, page 236](#).

Notes on Usage

- For general information on Processing-on-the-fly correction and determining scaling factors, see [Chapter 8.6 "Processing-on-the-fly", page 241](#).
- 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 be 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 has been 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 2D 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 a Processing-on-the-fly correction with positional values via `McBSP`.
- Conversely, Processing-on-the-fly correction activated by `set_fly_z` gets deactivated by such a correction. 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).
- `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 294](#). You can use `clear_fly_overflow` and `clear_fly_overflow_ctrl` to reset the error bits (Bit #24...Bit #25 from `get_marking_info`) for customer-specific monitoring of FlyZ applications.
- A 3D Processing-on-the-fly correction with positional values via `McBSP` can be activated by `set_multi_mcbsp_in` and `set_multi_mcbsp_in_list`.

8.6.12 "Fly Extension" Commands

- In order to be able to use the Processing-on-the-fly functionality flexibly and generically, the "Fly Extension" Commands are available in RTC6 Software Package \geq V1.6.1 (DLL 617, OUT 617, RBF 623), see Table 2.

Table 2: Number of supported axes in "Fly Extension" Commands.

"Fly Extension" Command ^(a)	Axes
<code>set_fly_1_axis</code>	1
<code>fly_return_1_axis</code>	1
<code>activate_fly_1_axis</code>	1
<code>park_position_1_axis</code>	1
<code>park_return_1_axis</code>	1
<code>wait_for_1_axis</code>	1
<code>set_fly_2_axes</code>	2
<code>fly_return_2_axes</code>	2
<code>activate_fly_2_axes</code>	2
<code>park_position_2_axes</code>	2
<code>park_return_2_axes</code>	2
<code>wait_for_2_axes</code>	2
<code>set_fly_3_axes</code>	3
<code>fly_return_3_axes</code>	3

(a) Only list commands, no control commands.

- The Processing-on-the-fly functionality⁽¹⁾ itself remains unchanged. Only its control has been made more flexible.
- "Fly Extension" Commands are a flexible system of Processing-on-the-fly applications.
 - Most of them affect:
 - only 1, only 2 or all 3 spatial axes
 - alternatively 1 Rotary axis
 - At the same time the following can be freely assigned (Table 4)
 - Encoder counter "Encoder0" or "Encoder1"
 - and/or an McBSP input on each of these axes, even mixed

(1) As described in Chapter 8.6.1 "Intended Use and Initialization", page 241...Chapter 8.6.11 "Processing-on-the-fly Correction for the z Axis", page 256.

- Important: "Fly Extension" Commands are in no case compatible with the "Classic" Processing-on-the-fly commands⁽²⁾.
- A Processing-on-the-fly application with "Classic" Processing-on-the-fly commands⁽²⁾ must have been explicitly been terminated, before the functionality of one of the "Fly Extension" Commands can be activated, and vice versa.
- With "Fly Extension" Commands, axes are specified according to Table 3.

Table 3: Parameter Axis of the "Fly Extension" Commands.

Axis	Parameter value	Axis
1		x axis
2		y axis
3		z axis
4		Rotary axis

- With the "Fly Extension" Commands, linear axes 1, 2 and 3 cannot be combined simultaneously with a Rotary axis (4).
- "Fly Extension" Commands support 1, 2 or 3 Axes, see Table 2.

(2) See Footnote, page 241.

Notes on How to Use

“Fly Extension” Commands

- `set_fly_1_axis(Axis, Mode, Scale)` replaces one of the “Classic” Processing-on-the-fly commands:
 - `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_2_axes` combines the activation of any 2 linear axes and replaces for example, even `set_fly_2d`.
- `fly_return_1_axis`, `fly_return_2_axes` and `fly_return_3_axes` extend the functionality of the “Classic” Processing-on-the-fly commands `fly_return` and `fly_return_z`.
- `activate_fly_2_axes` replaces the “Classic” Processing-on-the-fly commands
 - `activate_fly_2d`
 - `activate_fly_2d_encoder`
 - `activate_fly_xy`
 - `activate_fly_xy_encoder`
- `wait_for_1_axis` can “wait” for any encoder value or a value transmitted by McBSP, thus substituting `wait_for_encoder_mode` and `wait_for_mcbsp`.
- `wait_for_2_axes` stands for `wait_for_encoder_in_range` and `wait_for_encoder_in_range_mode` and are able to wait for McBSP values in addition.
- The laser control (switch off the laser or leave it unchanged) can be set individually at `wait_for_1_axis (wait_for_2_axes)` with parameter `LaserMode (LaserMode)`.

- The galvanometer scanner movement (stand still or move along) can be specified with `wait_for_1_axis (wait_for_2_axes)` by parameter `WaitMode (WaitMode)`. Thus, it is no longer bound to `set_fly_2d` and `set_fly_x/set_fly_y`.
- “Fly Extension” Commands are also compatible with McBSP transmissions from `set_mcbsp_in` and `set_multi_mcbsp_in`. The Processing-on-the-fly corrections can be subsequently overwritten, for example, by Encoder-based corrections. However, the McBSP transmission to the memory locations themselves cannot be changed.
- “Fly Extension” Commands are – like “Classic” Processing-on-the-fly commands⁽¹⁾ – not compatible with an Online Positioning.
- “Fly Extension” Commands are able to “wait” for McBSP values, see Table 4.
- Modes and Encoders supported with “Fly Extension” Commands are shown in Table 4.

(1) See Footnote on [page 258](#).

Table 4: Supported Modes and Encoders with "Fly Extension" Commands.

Mode (parameter value <code>Mode</code>)	Encoder ("source")	Effect
0	Reserved.	Generates <code>get_last_error</code> return code <code>RTC6_PARAM_ERROR</code> .
1	Encoder0.	Only for Processing-on-the-fly-correction. Encoder counter "Encoder0"-scaled. <code>PreviewTime-corrected</code> .
2	Encoder1.	Only for Processing-on-the-fly-correction. Encoder counter "Encoder1"-scaled. <code>PreviewTime-corrected</code> .
3	Encoder2.	Only for Processing-on-the-fly-correction. Tied to encoder counter "Encoder0"-scaled. <code>PreviewTime-corrected</code> .
4	Encoder2.	Only for Processing-on-the-fly-correction. Tied to encoder counter "Encoder1"-scaled. <code>PreviewTime-corrected</code> .
5	Reserved.	Generates <code>get_last_error</code> return code <code>RTC6_PARAM_ERROR</code> .
6	McBSP 32 Bit.	Processing-on-the-fly correction with position specifications, <code>read_mcbsp(0)</code> .
7	McBSP 31 Bit.	<code>set_mcbsp_in</code> (<code>Mode</code> 1, 2 or 4) and <code>read_mcbsp(0)</code> .
8	McBSP 29 Bit.	Memory location for the x coordinate of the Processing-on-the-fly application from <code>set_multi_mcbsp_in</code> , <code>read_multi_mcbsp(0)</code> .
9	Reserved.	Generates <code>get_last_error</code> return code <code>RTC6_PARAM_ERROR</code> .
10	McBSP LOW 16 Bit.	Processing-on-the-fly correction with position specifications at 2 <code>Axes</code> .
11	McBSP LOW 15 Bit.	From <code>set_mcbsp_in</code> (<code>Mode</code> = 3).
12	McBSP 29 Bit.	Memory location for the y coordinate of the Processing-on-the-fly application from <code>set_multi_mcbsp_in</code> , <code>read_multi_mcbsp(1)</code> .
13	Reserved.	Generates <code>get_last_error</code> return code <code>RTC6_PARAM_ERROR</code> .
14	McBSP HIGH 16 Bit.	Processing-on-the-fly correction with position specifications at 2 <code>Axes</code> .
15	McBSP HIGH 15 Bit.	From <code>set_mcbsp_in</code> (<code>Mode</code> = 3).
16	McBSP 29 Bit.	Memory location for the z coordinate of the Processing-on-the-fly application from <code>set_multi_mcbsp_in</code> , <code>read_multi_mcbsp(2)</code> .
17	Encoder0.	Only for <code>wait_for_1_axis</code> and <code>wait_for_2_axes</code> . Not for Processing-on-the-fly corrections. Encoder counter "Encoder0"-unscaled and <code>PreviewTime-corrected</code> .
18	Encoder1.	Only for <code>wait_for_1_axis</code> and <code>wait_for_2_axes</code> . Not for Processing-on-the-fly corrections. Encoder counter "Encoder1"-unscaled and <code>PreviewTime-corrected</code> .
19	Encoder0.	Like 17, but not <code>PreviewTime-corrected</code> .
20	Encoder1.	Like 18, but not <code>PreviewTime-corrected</code> .



Notes on Table 4

- Each of these sources can be recorded by `set_trigger`/`set_trigger4`.
- **Mode 1...4** as well as **17...18** are also suitable for scan systems with SCANahead control.
- With `wait_for_1_axis` and `wait_for_2_axes`, **Mode 17...20** must be specified but not **1...4**.
- With the other “**Fly Extension**” Commands where an Encoder specification is possible, **Mode 1...4** must be specified but not **17...20**.
- With the **McBSP** modes:
 - Users must make sure that the data is actually transferred in the correct format
 - Users must additionally set (exception:
Mode = 6) a corresponding Processing-on-the-fly correction in `wait_for[*]` commands (by `set_fly[*]`, `set_mcbsp_in` or `set_multi_mcbsp_in`)
 - the RTC6 board “waits” for the Encoder values even without explicit activation of the Processing-on-the-fly correction
- A `get_last_error` return code `RTC6_PARAM_ERROR` is generated, if a **Mode** is not allowed.
- Further details can be found in the respective command descriptions.

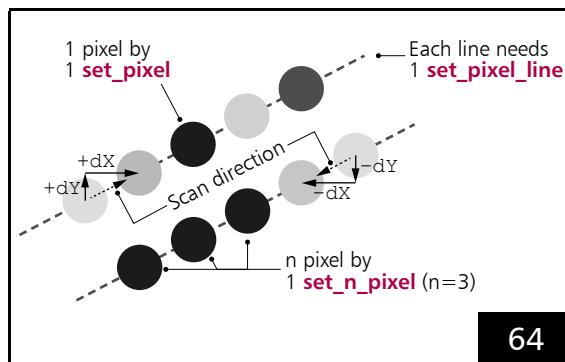
8.7 Pixel Output Mode

The **Vector commands** described in [Chapter 7.1 "Marking Dots, Lines and Arcs"](#), page 135 are intended for scanning vector based images. Furthermore, the RTC6 PCIe Board allows marking pixel images (bitmaps). With suitably adjusted lasers, black-and-white images or greyscale images can be obtained. Raster and vector based images can be combined as desired.

8.7.1 Principle of Operation

Pixel images are created image line by image line, see [Figure 64](#). Each line consists of a number of equidistant pixels. A line is generated in a single pass. During this pass, the laser focus moves – as with a normal **[*]mark[*] Command** – at an (approximately) constant velocity along the entire image line (the motion is split-up into **Microsteps**). The individual pixels are marked in passing: each pixel gets a laser pulse assigned at the appropriate location. By varying the laser energy from pixel to pixel, greyscale images (including black & white images) are produced.

Different output ports can be used to control the laser depending on mode and pixel output frequency, see command descriptions.



An individual **set_pixel_line** is required for each image line. The individual pixels of this line are then defined by successive **set_pixel**/**set_n_pixel**.

- By **set_pixel_line**, a single line is configured:
 - The *spatial* distance between the individual pixels with the parameters **dX**, **dY** (with **set_pixel_line_3d** additionally **dZ**)
 - The *temporal* distance between the individual pixels with the parameter **HalfPeriod**
 - The parameter **Channel** is extended compared to the RTC5 to **Channel** = **Mode** + **Port**, where now the (pixel) mode is defined via the value **Mode** and the **Output Port** is defined via the value **Port**, see [Table 5](#) and [6](#)

Notes

- The **Pixel Output Mode** can be combined with **Processing-on-the-fly**, see [Chapter 8.6 "Processing-on-the-fly"](#), page 241.
- The **Pixel Output Mode** *should not* be used in conjunction with "Automatic Laser Control" (see [Chapter 7.4.9 "Automatic Laser Control"](#), page 196), if there is a readjustment of the port output, pulse length (**PulseLength**) or output period (**HalfPeriod**) of the laser control signals **LASER1** and **LASER2**.
- The **Pixel Output Mode** *cannot* be combined with **Sky Writing**.⁽¹⁾
- The **Pixel Output Mode** *cannot* be combined with **Wobbel Mode**.

8.7.2 RTC6 Commands

Before writing an image line, a jump to the beginning of the line should be executed by a **Jump command**.

At the beginning of each image line, the **Pixel Output Mode** is activated by **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.

(1) However, the **Pixel Output Mode** can also be executed so that automatically generated (**Sky Writing Mode 1**-like) pre and post movements are included. This makes it easier to output pixel images with accurate positioning, see [Chapter 8.7.4 "Synchronization"](#), page 267.

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`) with `set_pixel_line`
- 3D vector (`dx,dy,dz`) with `set_pixel_line_3d`

Directly after `set_pixel_line`/`set_pixel_line_3d`, `set_pixel` has to be called separately for each pixel of the line. This defines the laser energies to be outputted at the corresponding pixel locations.

Pixel pulses are always outputted at the LASER1 output port, even for the purpose of synchronizing the laser (gating). If `PulseLength` is not specified as the port, the very `PulseLength` is outputted that has been specified at last by `set_laser_pulses`, however, at least $1/64 \mu\text{s}$.

As an alternative to a sequence of `n` identical `set_pixel` calls, `set_n_pixel` can be used.

Then only one command is stored on the RTC6 PCIe Board, but during list processing the corresponding `set_pixel` is executed `n` times. Particularly for black & white images, this can drastically reduce the size of lists. Do not confuse `set_n_pixel` with `n_set_pixel` (multi-board version of `set_pixel`).

Prior to the end of an image line, no command other than `set_pixel` or `set_n_pixel` must be inserted 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 [Section "Normal, Short, Variable and Multiple List Commands", page 301](#).

Notes

- The number of pixels in an image line is limited only by the capacity of the RTC6 PCIe Board list memory, see [Chapter 15 "Technical Specifications – RTC6 PCIe Board", page 854](#). 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 `set_pixel_line` or `set_pixel_line_3d`.
- The pixel distance (`dx, dy`) in the x direction and y direction (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.
- Depending on the [Pixel Output Mode](#), the half pixel output period can be any integer-multiple of $1/64 \mu\text{s}$. It is independent from the $10 \mu\text{s}$ position output period of the galvanometer scanners' (split-up into [Microsteps](#)) motion. Very low pixel output frequencies result in multiple galvanometer scanner steps per pixel, higher frequencies result in multiple pixel pulses per galvanometer 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 [micro_vector\[*\] commands](#), see [Chapter 8.8 "micro_vector\[*\] Commands", page 272](#).

8.7.3 Laser Control

Depending on the type of laser employed, the laser energy outputted at each pixel position can be varied by:

- laser pulse length
- laser power per pixel

“Classic Mode”

The “Classic Mode” (Mode = 0, see [Channel](#)) corresponds to the **Pixel Output Mode** of the RTC4 and RTC5. Maximum pixel output frequency: 400 kHz.

- Variation Of Laser Pulse Length: **set_pixel** defines – for each pixel – the length of the pixel pulse (in units of 1/64 μ s), which is outputted at the LASER1 port. For synchronization, see [Chapter 8.7.4 “Synchronization”, page 267](#).
- Variation Of Laser Power: **set_pixel** allows specification of a 12-bit analog voltage level for each pixel. The value is transferred either to the **ANALOG OUT1** or **ANALOG OUT2** output port (see above). For synchronization, see [Chapter 8.7.4 “Synchronization”, page 267](#). **set_n_pixel** defines the analog voltage level for n directly successive pixels of an image line.

“Extended Mode”

In “Extended Mode” (Mode = 16, see [page 703](#)), two pixels (32-bit values) are outputted at the specified port with each **set_pixel**. Maximum pixel output frequency: 800 kHz.

“Fast Mode”

In “Fast Mode” (Mode = 32, see [page 703](#)), 4 pixels of 16 bits each are outputted at the specified port with each **set_pixel**. Maximum pixel output frequency: 1.6 MHz.

“Ultra Fast Mode”

In “Ultra Fast Mode” (Mode = 64, see [page 703](#)) 8 pixels of 8 bits each are outputted at the specified port with each **set_pixel**. Maximum pixel output frequency: 3.2 MHz.

Notes

- Pixel output frequencies > 800 kHz (“Fast Mode” and “Ultra Fast Mode”) require the **Option “UFPM”**. Otherwise, the pixel output frequency is clipped to 800 kHz.
- It is recommended that some experiments be performed to determine an appropriate gradation curve for producing smooth greyscales. The resulting pixel greyscale values (“colors”) strongly depend on the employed material and the laser.
- The LASERON signal – as with a normal **[*]mark[*] Command** – switches to “On” after the **LaserOn Delay** has expired and remains “On” for the entire pixel line. After the last pixel has been outputted, it automatically goes to “Off”. See [Chapter 7.4 “Laser Control”, page 183](#).
- The LASER1 signal depends on the respective laser mode and the associated settings, as well as on the Mode (see [page 703](#)) set by **set_pixel_line/ set_pixel_line_3d**: A periodic signal with **HalfPeriod** from **set_pixel_line** and a **PulseLength** from **set_laser_pulses** (however, at least 1/64 μ s long) is outputted. The following exceptions apply:
 - In **Laser Mode 4** and **Laser Mode 6**, only a standby signal is outputted that cannot be varied. Power changes are only possible via the output ports **ANALOG OUT1**, **ANALOG OUT2**, 8-bit digital output port (EXTENSION 2) or 16-bit digital output port (EXTENSION 1).
 - If an output to **PulseLength** is specified as the Mode, **PulseLength** is varied per pixel.
 - No Softstart is performed, see [Chapter 7.4.7 “Softstart Mode \(not yet implemented\)”, page 194](#).
- The LASER2 signal follows the specifications of the respective laser mode.
- **Pixel Output Mode** and **Sky Writing** cannot be combined. However, a separate **Sky Writing Mode 1**-like galvanometer scanner movement can be inserted, see [Figure 65](#).
- The values for **HalfPeriod** and **PulseLength** set prior to **set_pixel_line** are not restored again. **set_laser_pulses** must be explicitly called again.

Table 5:

Mode	Max. frequency	No. of pixels per <code>set_pixel</code> / <code>set_n_pixel</code>	1 st parameter of <code>set_pixel</code> / <code>set_n_pixel</code> (name: <code>PortOutValue1</code> ^(a))	2 nd parameter of <code>set_pixel</code> / <code>set_n_pixel</code> (name: <code>PortOutValue2</code> ^(b))
0, 256 ^(c)	400 kHz	1	Pixel 1 (defined by the pulse length). The output is done as signal LASER1.	Port output (but not port 5) for Pixel 1 Pixel 1 (1 st parameter: 32-bit; 2 nd parameter: up to 16 bit, depending on port)
16	800 kHz	2	Port output for Pixel 1 (up to 32 bits per pixel, depending on port)	Port output for Pixel 2 (up to 32 bits per pixel, depending on port) Pixel 1 Pixel 2
32 ^(d)	1.6 MHz	4	Port output for Pixel 1, 2 (up to 16 bits per pixel, depending on port)	Port output for Pixel 3, 4 (up to 16 bits per pixel, depending on port) Pixel 2 Pixel 1 Pixel 4 Pixel 3
64 ^(d)	3.2 MHz	8	Port output for Pixel 1, 2, 3, 4 (up to 8 bits per pixel, for all ports)	Port output for Pixel 5, 6, 7, 8 (up to 8 bits per pixel, for all ports) Pixel 4 Pixel 3 Pixel 2 Pixel 1 Pixel 8 Pixel 7 Pixel 6 Pixel 5
			Bit 31.....Bit 0	Bit 31.....Bit 0

(a) `PulseLength` in RTC6 Software Packages < v1.3.3.

(b) `AnalogOut` in RTC6 Software Packages < v1.3.3.

(c) The galvanometer scanner movement is *not* continuous in Mode = 0 and continuous in Mode = 256.

(d) > 800 kHz only possible with **Option "UFPM"**.



Table 6:

Output Port	Where at the RTC6 PCIe Board	
1	12-bit analog output port 1	LASER Connector, page 72, ANALOG OUT1 See Figure 17 .
2	12-bit analog output port 2	LASER Connector, page 72, ANALOG OUT2 See Figure 17 . MARKING ON THE FLY Socket Connector, page 81, ANALOG OUT2 See Figure 25 .
3	8-bit digital output port	EXTENSION 2 Socket Connector, page 79 See Figure 24 .
4	16-bit digital output port	EXTENSION 1 Socket Connector, page 77 See Figure 22 .
5 ^{(a)(b)}	Pulse length	LASER Connector, page 72, LASER1 See Figure 17 .

(a) Port = 5 cannot be combined with Mode = 0.

(b) Port = 5 cannot be combined with Mode = 256.

8.7.4 Synchronization

The pixel output timing diagram for one image line with 3 pixels is shown in [Figure 66](#).

To prepare the laser control, `set_pixel_line/set_pixel_line_3d` switches off the Signals for "Laser Active" Operation from a previous Mark command after a [LaserOff Delay](#) (as with a [Jump command](#)) and waits until the laser is actually off.

During this waiting period, the galvanometer scanners do not move in "Classic Mode" (Mode = 0, see [page 703](#)) only. In all other modes, they continue to move at the speed defined by `HalfPeriod` and pixel distance in equidistant [Microsteps](#). This allows to program jerk-free run-in and run-out movements. Initial acceleration phases can be hidden by a [LaserOn Delay](#) (as with a normal `[*]mark[*]` Command) or by a corresponding number of "idle pixels" (see below).

After that, depending on the laser mode, pixel output starts immediately or after a Q-Switch delay, see [Figure 66](#). Analog signals at `ANALOG OUT1` or `ANALOG OUT2` change synchronously with the leading edge of each pixel pulse. The digital-to-analog converter requires about $1.5 \mu\text{s} \dots 3 \mu\text{s}$ to produce a stable analog output signal. With pixel output frequencies above around 100 kHz (`HalfPeriod` < approx. 320) digital-to-analog conversion cannot always be fully completed. At such pixel output frequencies, it must be carefully checked whether the functionality is sufficient for the intended purpose.

Bit #1 in `set_laser_control(Ctrl)` can be used to shift the laser pulse and thus the start of the digital-to-analog conversion by half a pixel period.

The pixel line ends with the first list command that is not a `set_pixel` or `set_n_pixel`.

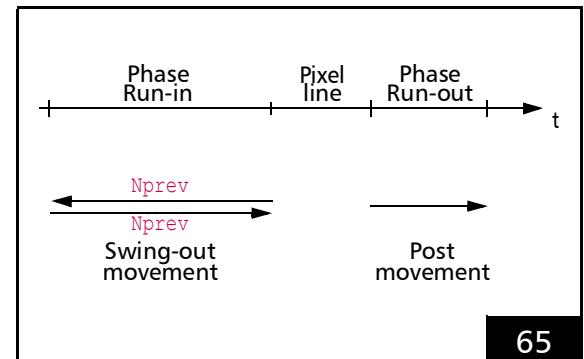
For the laser to be switched off even in the middle of a $10 \mu\text{s}$ cycle, a default pixel is automatically outputted after the last pixel. This is repeated as often as necessary until a started $10 \mu\text{s}$ cycle is finished. Then the laser is finally switched off.

The default pixel should be defined appropriately prior to `set_pixel_line/set_pixel_line_3d` in order to achieve a non-visible laser marking (= "idle pixels") in the run out, see `set_port_default` and `set_default_pixel`.

The RTC6 board waits – especially at pixel output frequencies $< 100 \text{ kHz}$ – until the default pixel is outputted.

The galvanometer scanners continue to run during this time, to ensure jerk-free connection movements (programmed by the user). No scanner delay is automatically inserted. In "Classic Mode" (Mode = 0, see [page 703](#)), the galvanometer scanners remain idle for a few clock cycles. With Mode + 256 this can be suppressed (even for the forerun phase).

The [Tracking error](#) and the hidden acceleration phase mean a pixel line shift", see [Figure 66](#). Normally, this needs to be compensated for by an adjusted run-in. To make this easier, Mode + 512 can be used to switch on [Sky Writing Mode 1](#)-similar movements in the run-in and run-out phase and thus place the pixel line with pinpoint accuracy, see [Figure 65](#).



65

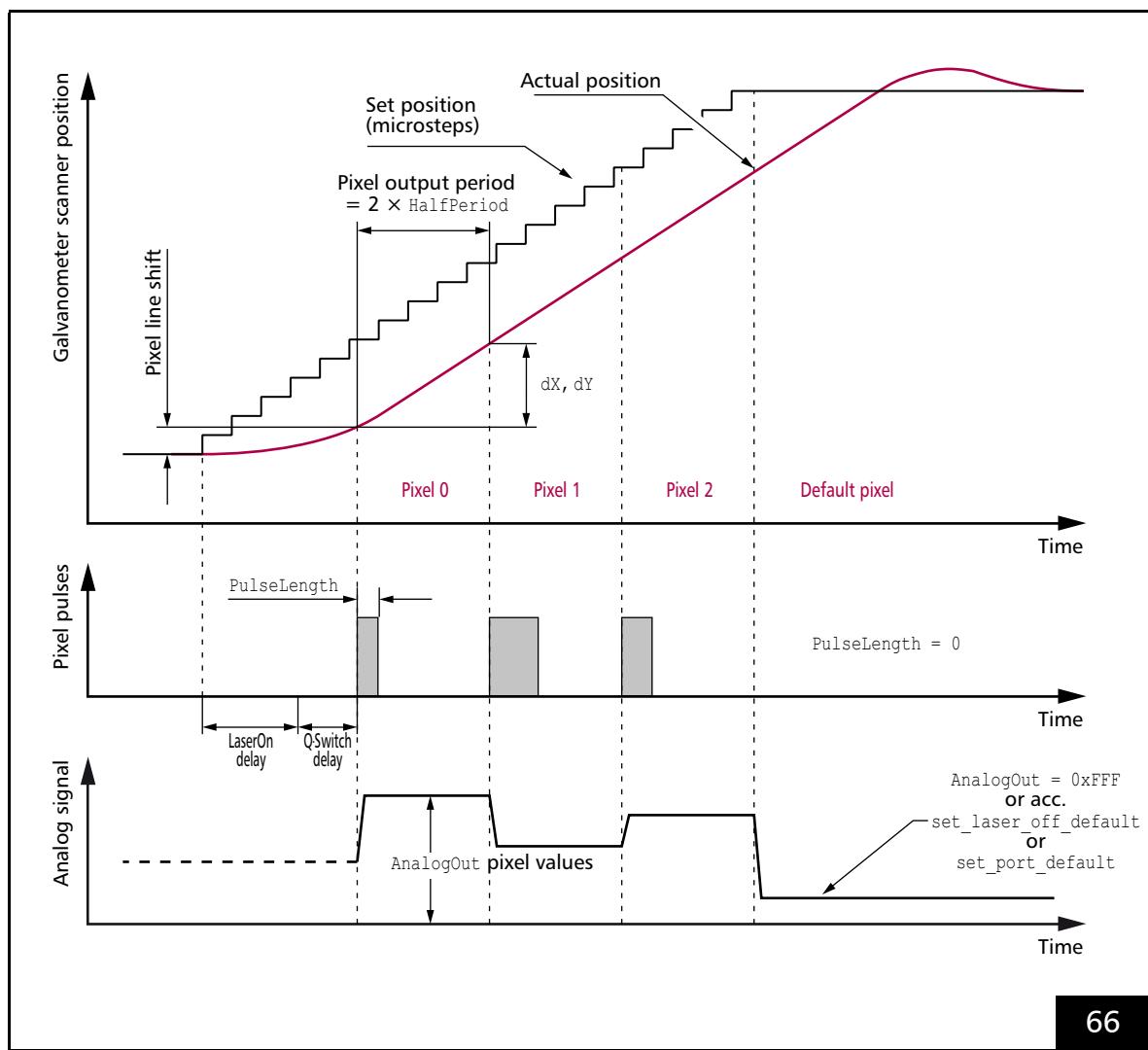
Modus + 512

These movements cannot be combined with [Sky Writing Mode 2](#). The duration of the swing-out movement must be explicitly defined via `Nprev` of `set_sky_writing_para` in advance.⁽¹⁾

(1) With excelliSCAN scan heads, [Sky Writing](#) and automatic delay calculation need to be switched on, see "excelliSCAN Scan Heads – Functional Principle of SCANAhead Servo Control and Operation by RTC6 Boards" Manual.

Notes

- With $\text{HalfPeriod} < \text{PulseLength} / 2$ the laser does not switch off between the pixels.
- When specifying `HalfPeriod` and pixel distance, observe the dynamics of the scan system (mark speed) and the properties of the laser system (power modulation).



Timing of the galvanometer scanner positions and the laser control signals in the **Pixel mode** $\text{Mode} = 0$ and a **YAG Mode**, see [Chapter 7.4.4 "YAG Modes 1, 2, 3, 5"](#), page 189.

The pixel output period in this example is approx. 4.5 **Microsteps**.



Example Code

The example code must be included in a user program.

```

//  UINT = uint32_t
//  LONG = int32_t

// laser parameters
UINT LaserMode = 0;
UINT LaserControl = 0x0;
LONG LaserOnDelay = (LONG)round(100.0*64.0); // LaserOnDelay = 100 µs
UINT LaserOffDelay = (UINT)round(100.0*64.0); // LaserOffDelay = 100 µs
UINT HalfPeriod = (UINT)round(5.0 * 32.0); // Period = 5 µs, PixelFrequency = 200 kHz
UINT PulseLength = (UINT)round(0.5 * 64.0); // PulseLength = 0.5 µs

// Pixel Output Mode variables
enum E_PixelModes
{
    STANDARD      = 0,      // like RTC5-Mode; up to 400 kHz; 1 pixel per set_pixel command; 2 values of 32 bit per pixel
    ENHANCED      = 16,     // up to 800 kHz; 2 pixel per set_pixel command; 1 value of 32 bit per pixel
    FAST          = 32,     // up to 1.6 MHz; 4 pixel per set_pixel command; 1 value of 16 bit per pixel
    ULTRAFAST     = 64,     // up to 3.2 MHz; 8 pixel per set_pixel command; 1 value of 8 bit per pixel
    STANDARD_MOVE = 256    // like STANDARD, but with continuous galvanometer scanner movement
} Mode;

enum E_PixelPorts
{
    NO_OUT_PORT    = 0,
    ANALOG_OUT1    = 1,      // LASER Connector ANALOG OUT1
    ANALOG_OUT2    = 2,      // LASER Connector ANALOG OUT2
    DIGITAL_8Bit    = 3,      // EXTENSION 2 socket connector (8-bit)
    DIGITAL_16Bit   = 4,      // EXTENSION 1 socket connector (16-bit)
    PULSE_LENGTH    = 5      // not allowed for mode = 0 or mode = 256
} Port;

Port = DIGITAL_8Bit;           // pixel output port number
Mode = STANDARD;              // Pixel Output Mode
UINT Channel = Port + Mode;   // pixel channel

UINT Number = 1;               // set_pixel repetition number

UINT PixelArray[17] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 }; // port output values
UINT PortOutValue1 = PixelArray[0];
UINT PortOutValue2 = PixelArray[0];

LONG dx = (LONG)round(0.01 * CalibrationFactorXY); // 10 um spot distance in x direction
LONG dy = (LONG)round(0.0 * CalibrationFactorXY); // 0 um spot distance in y direction

LONG PixelLineStartPosX = (LONG)round(0.0 * CalibrationFactorXY);
LONG PixelLineStartPosY = (LONG)round(0.0 * CalibrationFactorXY);

// basic configuration
set_laser_mode(LaserMode);
set_laser_control(LaserControl);

// RTC-List
set_start_list_pos(1, 0);
set_laser_delays(LaserOnDelay, LaserOffDelay);
set_default_pixel_list(0); // sets pixel default PulseLength
UINT PortDefault = Port - 1; // CAUTION: port numbers differ between set_port_default/set_port_default_list and
                           // set_pixel_line
set_port_default_list(PortDefault, PixelArray[0]); // sets default value for specified port

```



```

switch (Mode)
{
case STANDARD:
case STANDARD_MOVE:
{
    // standard Pixel Output Mode up to 400 kHz (Mode = 0, 256)
    // 2 values of 32 bit per pixel
    // 1 pixel per set_pixel command
    //-----
    // |      pixel1      |
    // | 32 Bit  || 32 Bit  |
    // | PulseLength || PortOutValue |
    //-----
    HalfPeriod = (UINT)round(5.0 * 32.0);    // Period = 5 µs, PixelFrequency = 200 kHz
    PulseLength = (UINT)round(0.5 * 64.0);    // PulseLength = 0.5 µs

    jump_abs(PixelLineStartPosX, PixelLineStartPosY);
    set_pixel_line(Channel, HalfPeriod, dx, dy);
    set_n_pixel(1 * PulseLength, PixelArray[1], Number);           // pixel 1
    set_n_pixel(2 * PulseLength, PixelArray[2], Number);           // pixel 2
    set_n_pixel(3 * PulseLength, PixelArray[3], Number);           // pixel 3
    set_n_pixel(4 * PulseLength, PixelArray[4], Number * 2);       // (pixel 4)*2

}
break;

case ENHANCED:
{
    // enhanced Pixel Output Mode up to 800 kHz (Mode = 16)
    // 1 value of 32 bit per pixel
    // 2 pixel per set_pixel command
    //-----
    // |      pixel1      ||      pixel2      |
    // | 32 Bit  || 32 Bit  |
    // | PortOutValue1 || PortOutValue2 |
    //-----
    HalfPeriod = (UINT)round(2.0 * 32.0);    // Period = 2 µs, PixelFrequency = 500 kHz
    PulseLength = (UINT)round(0.5 * 64.0);    // PulseLength = 0.5 µs
    set_laser_pulses(HalfPeriod, PulseLength);

    jump_abs(PixelLineStartPosX, PixelLineStartPosY);
    set_pixel_line(Channel, HalfPeriod, dx, dy);
    set_n_pixel(PixelArray[1], PixelArray[2], Number);           // pixel 1, pixel 2
    set_n_pixel(PixelArray[3], PixelArray[4], Number);           // pixel 3, pixel 4
    set_n_pixel(PixelArray[5], PixelArray[6], Number * 2);       // (pixel 5, pixel 6)*2
    //--> pixel output: pixel 5, pixel 6, pixel 5, pixel 6

}
break;

```



```

case FAST:
{
    // fast Pixel Output Mode up to 1.6 MHz (Mode = 32) - needs UFPM-Option
    // 1 value of 16 bit per pixel
    // 4 pixel per set_pixel command
    //-
    // |  pixel2  |  pixel1  ||  pixel4  |  pixel3  |
    // |  16 Bit  |  16 Bit  ||  16 Bit  |  16 Bit  |
    // |  PortOutValue1  ||  PortOutValue2  |
    //-
    HalfPeriod = (UINT)round(1.0 * 32.0);    // Period = 1 µs, PixelFrequency = 1 MHz
    PulseLength = (UINT)round(0.5 * 64.0);    // PulseLength = 0.5 µs
    set_laser_pulses(HalfPeriod, PulseLength);

    jump_abs(PixelLineStartPosX, PixelLineStartPosY);
    set_pixel_line(Channel, HalfPeriod, dx, dy);
    // low 16 bit - first pixel, high 16 bit - second pixel
    PortOutValue1 = PixelArray[1] | (PixelArray[2] << 16);
    PortOutValue2 = PixelArray[3] | (PixelArray[4] << 16);
    set_n_pixel(PortOutValue1, PortOutValue2, Number);           // pixel 2 + pixel 1, pixel 4 + pixel 3

    PortOutValue1 = PixelArray[5] | (PixelArray[6] << 16);
    PortOutValue2 = PixelArray[7] | (PixelArray[8] << 16);
    set_n_pixel(PortOutValue1, PortOutValue2, Number * 2);      // (pixel 6 + pixel 5, pixel 8 + pixel 7)*2
    //--> pixel output: pixel 5, pixel 6, pixel 7, pixel 8, pixel 5, pixel 6, pixel 7, pixel 8

}
break;
case ULTRAFAST:
{
    // ultra fast Pixel Output Mode up to 3.2 MHz (Mode = 64) - needs UFPM-Option
    // 1 value of 8 bit per pixel
    // 8 pixel per set_pixel command
    //-
    // |  pixel4  |  pixel3  |  pixel2  |  pixel1  ||  pixel8  |  pixel7  |  pixel6  |  pixel5  |
    // |  8 Bit   |  8 Bit   |  8 Bit   |  8 Bit   ||  8 Bit   |  8 Bit   |  8 Bit   |  8 Bit   |
    // |  PortOutValue1  ||  PortOutValue2  |
    //-
    HalfPeriod = (UINT)round(0.5 * 32.0);    // Period = 0.5 µs, PixelFrequency = 2 MHz
    PulseLength = (UINT)round(0.25 * 64.0);    // PulseLength = 0.25 µs
    set_laser_pulses(HalfPeriod, PulseLength);

    jump_abs(PixelLineStartPosX, PixelLineStartPosY);
    set_pixel_line(Channel, HalfPeriod, dx, dy);

    // lowest 8 bit - first pixel, highest 8 bit - fourth pixel
    PortOutValue1 = PixelArray[1] | (PixelArray[2] << 8) | (PixelArray[3] << 16) | (PixelArray[4] << 24);
    PortOutValue2 = PixelArray[5] | (PixelArray[6] << 8) | (PixelArray[7] << 16) | (PixelArray[8] << 24);

    // pixel 4 + pixel 3 + pixel 2 + pixel 1, pixel 8 + pixel 7 + pixel 6 + pixel 5
    set_n_pixel(PortOutValue1, PortOutValue2, Number);

    PortOutValue1 = PixelArray[ 9] | (PixelArray[10] << 8) | (PixelArray[11] << 16) | (PixelArray[12] << 24);
    PortOutValue2 = PixelArray[13] | (PixelArray[14] << 8) | (PixelArray[15] << 16) | (PixelArray[16] << 24);

    // (pixel 12 + pixel 11 + pixel 10 + pixel 9, pixel 16 + pixel 15 + pixel 14 + pixel 13)*2
    set_n_pixel(PortOutValue1, PortOutValue2, Number * 2);
    // --> pixel output: pixel 9, pixel 10, pixel 11, pixel 12, pixel 13, pixel 14, pixel 15, pixel 16,
    //           pixel 9, pixel 10, pixel 11, pixel 12, pixel 13, pixel 14, pixel 15, pixel 16

}
break;
default:
break;
}

set_end_of_list();

```

8.8 `micro_vector[*]` Commands

`micro_vector[*]` commands move the galvanometer scanners directly to the specified position by a "Hard jump"

- in 2D Image field:
 - `micro_vector_abs`
 - `micro_vector_rel`
- in 3D image field:
 - `micro_vector_abs_3d`
 - `micro_vector_rel_3d`

The Signals for "Laser Active" Operation can be switched on and off individually for each `micro_vector[*]` command by the specified `Laser Delays` (parameters `LasOn` and `LasOff`). The `Laser Delays` defined by `set_laser_delays` are not overwritten by that. These remain valid for normal `Jump commands` and `[*]mark[*]` Commands.

The `micro_vector[*]` commands can be used to mark trajectories (see Glossary entry on [page 29](#)) of arbitrary shape and at variable speed, as long as the dynamic limits of the scan system are not exceeded.

Notes

- `micro_vector[*]` commands wait for a preceding scanner delay. They never trigger new `Scanner Delays`.
- Users themselves are responsible for appropriately parameterizing `micro_vector[*]` commands:
 - Complying with the dynamic limits of the scan system
 - Avoiding cross over and take over with `LaserON` and `LaserOff`, see [Chapter 7.2.3 "Notes on Optimizing the Delays", page 154](#)
- A Wobbel motion specified by `set_wobbel` or `set_wobbel_mode`, see [Chapter 8.4 "Wobbel Mode", page 231](#) is not taken into account (but also not deactivated).
- The `Sky Writing` mode is not taken into account (but also not deactivated).
- The output values are calculated according [Chapter 7.3.6 "Output Values to the Scan System", page 180](#) without 1 and 2.

8.9 Timed Commands

“Normal” **Vector commands** and “Arc” **commands** are executed in such a way that the laser focus moves with a defined speed⁽¹⁾. This is fine for most laser marking and laser material processing applications.

At the beginning of a jump, the **Signals for “Laser Active” Operation** are switched off. The **Signals for “Laser Standby” Operation** are switched on depending on the settings, see **Chapter 7.4.1 “Enabling, Activating and Switching Laser Control Signals”, page 183**.

However, some applications require that each **Vector command** or “Arc” **command** consumes exactly the same amount of time, regardless of its spacial length.

In this case, it is necessary to specify a jump *duration* or mark process *duration* (rather than the jump speed or mark speed).

Timed commands allow specification of the duration of the **Vector command** or “Arc” **command** with an accuracy of 10 μ s (the output period of the **Microsteps**) and in the range from 10 μ s to 167,772,160 μ s (\approx 2.8 min).

A vector or arc is split-up into the specified (T Parameter) number of **Microsteps**.

For $T \geq 5$, the following applies:

$$\text{Length } \Delta s \text{ of a Microstep} = L / t \\ \text{with } t = \text{integer}((T + 5) / 10)^{(2)}$$

Thus, jump speed and mark speed automatically depend on the length of the vectors or arcs.

The following timed commands are available:

- Vectors and arcs
 - **timed_jump_abs**
 - **timed_jump_rel**
 - **timed_mark_abs**
 - **timed_mark_rel**
 - **timed_arc_abs**
 - **timed_arc_rel**
- Parametrized vectors
 - **timed_para_jump_abs**
 - **timed_para_jump_rel**
 - **timed_para_mark_abs**
 - **timed_para_mark_rel**
- 3D vectors
 - **timed_jump_abs_3d**
 - **timed_jump_rel_3d**
 - **timed_mark_abs_3d**
 - **timed_mark_rel_3d**
- Parametrized 3D vectors
 - **timed_para_jump_abs_3d**
 - **timed_para_jump_rel_3d**
 - **timed_para_mark_abs_3d**
 - **timed_para_mark_rel_3d**

Notes

- After a timed command, a “normal” **Jump Delay**, **Mark Delay** or **Polygon Delay** is inserted.
- Ellipses are fundamentally not available as timed commands.
- With $T < 5 \mu$ s, a timed command is synonym to its “normal” (= without [**timed_***]) command.
- The total execution time of a timed command is the sum of the specified time and the associated delay, see also **Chapter 7.2.2 “Scanner Delays”, page 146**.

(1) Either jump speed or mark speed.

(2) For $T < 5$, the command is carried out as non-timed command, see **Chapter 7.1.2 “Microstepping”, page 139**.

8.10 Automatic Self-Calibration

8.10.1 Use for Drift Compensation

Long-term repeatability is very important in many applications, for example, 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 galvanometer scanners have reached their operating temperature. In addition, the magnitudes of environmental fluctuations (for example, 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 equipped with an additional internal sensor system for automatic self-calibration (ASC sensor system, Home-In sensors). This allows the position detectors of the galvanometer scanners to be calibrated and gain drift and offset drifts to be reliably compensated. The positioning accuracy is thus maintained over long periods of time. Remaining long-term drift effects are of the same order of magnitude as short-term repeatability accuracies.

8.10.2 How it Works

By **auto_cal**, a measurement routine can be started for determining the exact control values for reference positions (Home-In positions) defined by the internal sensor system.

For drift *measurement*, the routine should be executed:

- When setting up the equipment – to determine reference values for the Home-In positions, see [Chapter 8.10.3 "Determining Reference Values", page 276](#)
- During the application at appropriate time intervals – to determine if and how the Home-In positions have changed, see [Chapter 8.10.4 "Calibration During the Application", page 276](#)

For drift *compensation*, appropriate gain values 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. See also [Chapter 7.3.6 "Output Values to the Scan System", page 180, #9](#).

The setting can also be made manually with **set_hi**, if customer-specific measuring procedures are to be used, see [Section "Customer-Specific Calibration", page 277](#).

Notes

- Prior to performing a measurement routine, **auto_cal(Command = 4)** can be used to check if:
 - the attached scan system in fact has an internal sensor system for automatic self-calibration (Home-In sensors)
 - the sensor system is functioning properly

This ASC hardware check also occurs automatically by `auto_cal(Command = 0)` and if required, 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
 - no other commands should be sent to the scan system
- For Gain/Offset Correction:
See [Chapter 7.3.6 "Output Values to the Scan System", page 180 #9](#).
After an initialization by `load_program_file` or `auto_cal(Command = 2)`, the following is set:
 - Gain = 1.0
 - Offset = 0

- For **iDRIVE** scan systems⁽¹⁾, the following applies in addition:
 - At the beginning of a measurement routine, the **XY2-100 status word** is automatically set as to-be-returned by the scan system. At the end of the measurement routine, the previously set data type is restored.
 - `auto_cal` aborts with an error result value 7 if
 - `auto_cal` is to be executed for the first scan head connector, and
 - an "Automatic Laser Control" in `Mode = 2` (basic mode) has been activated, see [Section "Speed-Dependent Laser Control", page 200](#).
The "Automatic Laser Control" must be deactivated before performing automatic self-calibration.
 - `auto_cal` also aborts (result value 1, 10 or 11), if the scaling factor has been 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)`. The scaling factor must have been set to 1 (default setting) by prior to automatic self-calibration
 - `control_command(Data = 1283_H)` and
 - `control_command(Data = 1200_H)`.

(1) See Glossary entry on [page 26](#).

8.10.3 Determining Reference Values

The reference values are determined by `auto_cal` (`Command` = 0). This starts the measurement routine for determining the current Home-In positions. These are then the reference values for later calibrations with `auto_cal` (`Command` = 1 or 3). Immediately after determination, they can be read out by `get_hi_pos` and are available for customer-specific calibration, see [Section "Customer-Specific Calibration", page 277](#). The reference values are stored in the **Flash memory** of the RTC6 board. This guarantees that the reference values are available even after a restart.

Notes

- After `auto_cal` (`Command` = 0), the galvanometer scanners are in the same real **Image field** position as before the command.
- The reference values should be determined when the machine is set up, for example, when the scan system is installed or exchanged. When exchanging the scan system, for which reference values have already been determined earlier and read out by `get_hi_pos`, these reference values can be transferred directly to the RTC6 board by `write_hi_pos`.
- Reference values 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 values 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.
- Storing the reference values takes several ms and interrupts the 10 μ s clock cycle of the **DSP**. For scan systems with clock cycle monitoring (interlock), this should be temporarily deactivated.

8.10.4 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).

This starts a measurement routine for determining the current Home-In positions. From the deviations from the stored reference values (see [Chapter 8.10.3 "Determining Reference Values", page 276](#)) determined in the process, gain values and offset values are calculated and set separately for the x axis and y axis.

This drift compensation is effective immediately. It is valid until:

- `auto_cal` (`Command` = 1) is called again
- it is switched off
 - by `auto_cal` (`Command` = 2)
 - after `load_program_file`

Notes

- After `auto_cal` (`Command` = 1) the galvanometer scanners are in the same position in the real **Image field** as before the command.
- The calibration routine started with `auto_cal` (`Command` = 1) typically lasts 1...2 seconds (depending on the strength of the drift). An ASC hardware check is automatically performed, if `auto_cal` (`Command` = 0 or 4) has not been executed prior to the first time call of `auto_cal` (`Command` = 1). This extends the execution of `auto_cal` by a few seconds.

Customer-Specific Calibration

Automatic self-calibration is midpoint centered, that is, the **Image field** center remains stable.

If an alternative is preferred (for example, 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 Home-In positions determined by **auto_cal** (`Command` = 0) can be read out by **get_hi_pos**. From this, compensating gain values and offset values can be calculated and set by **set_hi**. The drift compensation set in this way is effective immediately.

Notes

- **get_hi_pos** returns the last measured Home-In positions. These are the stored Home-In reference values immediately after
 - **auto_cal**(`Command` = 0)
 - initialization by **load_program_file**
- After **set_hi**, a correction of the current position occurs 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.

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.
- The measurement routine determines the Home-In positions by several measurements. If deviations between the individual measurements are too large (maximum – minimum > 96 bits), the measurement routine aborts and an error 2 is returned. In this case, no reference values are

stored for the affected axis (`Command` = 0) and the gain and offset factors remain unchanged with (`Command` = 1)⁽¹⁾. Then **get_hi_pos** returns 0 instead of the faulty values.

- An error within an individual measurement cycle can be caused by a brief mechanical or electrical disturbance. 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 values and offset values until new correction values are determined by the next (successful) measurement cycle.

Significant spreading across several measurement cycles 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 (for example, signal faulty or not found) get permanently stored. 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.

(1) (`Command` = 0) always sets gain = 1.0 and offset = 0.

8.11 Camming

camming produces a marking that simulates the classic camshaft action of moving a valve tappet – or more generally a cam disk moving a lever. An example for a **Camming** process is shown in [Figure 67](#).

The galvanometer scanner motion here is a lever movement defined as a 2D curve. It is written in a list as a closed point-by-point sequence of **mark_abs** commands.

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** at the position **FirstPos + Index**, whereby **Index** =
 $\text{Round}((\text{EncoderCurrent} - \text{EncoderStart}) \times \text{Scale})$.
FirstPos is the first **mark_abs** command’s list position and **Scale** is a freely selectable scaling factor. **EncoderStart** gets automatically determined when **camming** is called. There is no automatic encoder reset. The first outputted point is always **Index** = 0.

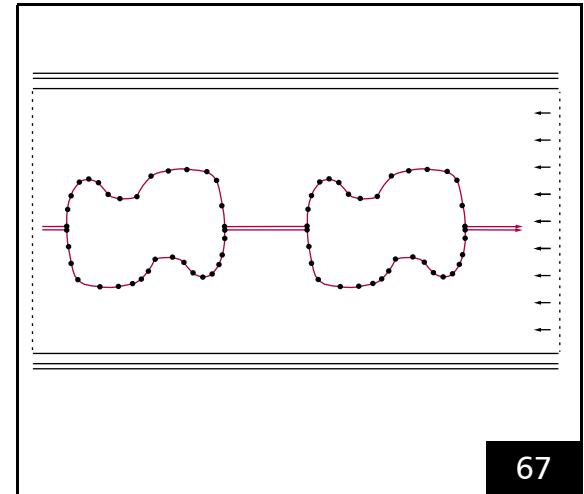
The individual points are executed every 10 μs as a “Hard jump” without **Scanner Delays**.

The distance between two points (= “resolution”) is freely selectable.

Scale determines 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 cycle and the spacing of the points determine the actual mark speed.

“Resolution”, **Scale** and encoder speed should suit the dynamic characteristics of the connected scan system.



67

Camming process 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:

- **Ctrl** > 0

The laser is controlled externally (with **laser_signal_on_list** before **camming** and **laser_signal_off_list** after **camming**)

- **Ctrl** = 0

The laser is controlled RTC6 board-internally automatically (as with a normal **Polyline** with consideration of **Laser Delays**)

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 accordance with encoder direction point lists can also run backward. Then, the curve is terminated with **Index** = 0.

The curve can be repeated indefinitely (**Ctrl** = 2 or **Ctrl** = 3). To avoid “Hard jumps” at the start of a repeat, the point list should represent a closed curve. Indefinite repeating must be cancelled by **stop_execution** or an external /STOP.

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

- An external `/START` during list execution is:
 - Suppressed during normal operation
 - Not suppressed during a **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 **camming** parameter `Scale`).

The laser control can be combined with the automatic ""vector-controlled 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.

In addition, a combination with Encoder-speed-dependent laser control is possible, see **Section "Encoder-Speed-Dependent Laser Control"**, page 203.

Alternatively, `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 cycle.

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 104), then the curve shape can be tested using a normal execution start, for example, by `execute_at_pointer(Pos)` (to some extend as a **Polyline**, but with a predefined mark 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 should include the "Hard jump", then `timed_mark_abs_3d` (with `Time` = 10 μ s) can be used as well, but not `timed_para_mark_abs_3d`.

8.12 Time Measurements

8.12.1 RTC6 Timer

RTC6 PCIe Boards are equipped with an integrated timer, which is referred to as the “RTC6 Timer” in the following. The **RTC6 Timer** only counts clock cycles of list commands. Counting is paused during interruptions by **set_wait** or **pause_list**.

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 **RTC6 Timer** value and resets it to 0. The elapsed time can then be read by **get_time**, which returns the **RTC6 Timer** value saved during the most recent call of **save_and_restart_timer**.

The present **RTC6 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 **RTC6 Timer** to zero. In this way the interim execution time of lengthy marking processes can be monitored.

8.12.2 Timestamps

32-bit “Timestamp Counter”

There is also a **32-bit “Timestamp Counter”** on RTC6 boards. It starts counting at 0 with **load_program_file** and counts uninterruptible all 10 μ s clock periods. Using the **32-bit “Timestamp Counter”**, an absolute reference to the individual time periods can be established, even if the **RTC6 Timer** has been reset by **save_and_restart_timer**. The **32-bit “Timestamp Counter”** can be recorded by **set_trigger/set_trigger4** (signal 52) and is read-out by **get_value** (52).

store_timestamp_counter / **store_timestamp_counter_list** reads out the **32-bit “Timestamp Counter”** from the RTC6 board and buffers it there as the time reference **TimeStampStorage**.

Subsequently, **wait_for_timestamp_counter** or **wait_for_timestamp_counter_mode** can be used to wait in a list for a fixed time difference to the stored **32-bit “Timestamp Counter”**. Overflows of the **32-bit “Timestamp Counter”** are detected and corrected.

About 12 hours is the longest waiting time that can be specified with the 32-bit offset. Waiting times beyond this must be broken down into shorter blocks and processed sequentially with sequences of **store_timestamp_counter_list** and **wait_for_timestamp_counter** / **wait_for_timestamp_counter_mode**. This is not always practicable with long process times.

64-bit “Timestamp Counter”

As of DLL 624, OUT 624 **get_timestamp_long** with **64-bit “Timestamp Counter”** and **wait_for_timestamp_counter_long** with 64-bit offset are available as an alternative. **get_timestamp_long** reads out the **64-bit “Timestamp Counter”**

(= **TimeStampCounterLong**) from the RTC6 board. Its lower part (= returned parameter value **TimeStampL**) and the **32-bit “Timestamp Counter”** are identical.

wait_for_timestamp_counter_long waits for an absolute **64-bit “Timestamp Counter”** value⁽¹⁾ (64-bit offset). The user program must control the synchronization.

Notes

- **get_time** and **get_lap_time** only take list execution times into account.
- To compare the RTC6 board-internal **save_and_restart_timer** time measurement to an external time measurement via the **BUSY** pin, you should insert a **list_nop** between **save_and_restart_timer** and **set_end_of_list**. This ensures that a 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).

(1) Analogously to **wait_for_timestamp_counter_mode**, however, **store_timestamp_counter** / **store_timestamp_counter_list** is not used.

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 (for example, workpiece placement, robotic motion, process monitoring etc.).

For this purpose, the RTC6 PCIe Board provides a variety of peripheral interfaces, see [Chapter 4.6 "Interfaces for the Laser and Peripheral Equipment", page 72](#).

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

9.1 Signal Output

For peripheral control (for example, 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 [Section "16-Bit Digital Input Port and 16-Bit Digital Output Port", page 77](#).

16-bit digital values are assigned to the output port by `write_io_port_list`, `write_io_port`, `write_io_port_mask_list`, `write_io_port_mask` or `write_port_list`. The output is in high-impedance mode until an initial value is assigned to it. In addition, `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:

- With position-dependent and speed-dependent laser control (see `set_port_default`)
- Upon terminating **Pixel mode**

If "Automatic Laser Control" is activated with `Ctrl=6` from `set_auto_laser_control`, then the value at the 16-bit digital output automatically gets adjusted, see [Chapter 7.4.9 ""Automatic Laser Control"", page 196](#). You can log this by `set_trigger/set_trigger4` (signal 24).

When the output value is outputted, 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...DATA7), see [Section "8-Bit Digital Output Port", page 80](#).

Its output values can be set by `write_8bit_port`, `write_8bit_port_list` or `write_port_list`. The output is in high-impedance mode until an initial value is assigned to it. In addition, `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:

- With position-dependent and speed-dependent laser control (see `set_port_default`)
- Upon terminating Pixel mode

If "Automatic Laser Control" is activated with `Ctrl = 3` from `set_auto_laser_control`, then the value at the 8-bit digital output automatically gets adjusted, see [Chapter 7.4.9 ""Automatic Laser Control""](#), page 196. This can be recorded by `set_trigger/set_trigger4` (signal 24).

When the output value is outputted, 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 [Section "8-Bit Digital Output Port", page 80](#).

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 [Section "2-Bit Digital Output Port", page 73](#).

By `set_laser_pin_out`, `set_laser_pin_out_list` or `write_port_list`, values are assigned to this output port.

The value is 0 (pins are LOW) until an initial value is assigned to it.

In addition, `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 is cancelled either by `stop_execution` or an external stop signal.

9.1.4 12-Bit Analog Output Port 1 and 2

The **LASER Connector** provides the two 12-bit analog output ports, **ANALOG OUT1** and **ANALOG OUT2**, see [Section "12-Bit Analog Output Port 1 and 2"](#), page 73.

ANALOG OUT2 is also available by the **MARKING ON THE FLY** socket connector, see [Section "Analog Output Port", page 81](#).

The output values can be separately set by `write_da_x`, `write_da_x_list` or `write_port_list`.

The values are 1 (corresponds to approx. 0 V) until initial values are assigned to it.

In addition, `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-dependent 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 [Chapter 7.4.7 "Softstart Mode \(not yet implemented\)"](#), page 194
- In Pixel Output Mode, see [Chapter 8.7 "Pixel Output Mode"](#), page 262

If "Automatic Laser Control" is activated with `Ctrl = 1` or `Ctrl = 2` from `set_auto_laser_control`, then the value at one of the 12-bit analog output ports automatically gets adjusted, see [Chapter 7.4.9 ""Automatic Laser Control""](#), page 196. This can be recorded by `set_trigger/set_trigger4` (signal 24).

9.1.5 Controlling Stepper Motors

Output Signals

The signals (ENABLE, DIRECTION and CLOCK) for controlling up to two stepper motors are outputted at the "STEPPER MOTOR" socket connector:

- 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 RTC6 PCIe Board 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 μ s 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 cycles 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 [Chapter 4.6.7 "STEPPER MOTOR Socket Connector", page 86](#).
- For querying signals, see [Section "Querying Signals and Status Values", page 285](#).
- Stepper motor signals are outputted independently of any executing lists. A `set_end_of_list`, `pause_list`, `set_wait`, `stop_execution` or external stops do not terminate or pause a forwarding 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.

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.

Once the reference movement has been successfully completed, the position variable (for the current position) is set to the value defined by parameter `Pos` as the reference for subsequent set-position movements. The reached reference position is offset from the limit switch position by $\pm \text{Tol}$ (direction dependent).

During reference movement, the status is "Init" (`Bit #5 = 1`), see [Section "Querying Signals and Status Values", page 285](#). The limit switch position is traversed 4 times and a mean limit switch position is calculated from this. Then the stepper motor moves away from the limit switch by `Tol` steps in the opposite direction to `stepper_init Dir`. `Tol` must be large enough to overcome a possible hysteresis. During this set-position movement, see [Section "Set-Position Movements", page 284](#), the status is no longer "Init" but "Busy" (`Bit #4 = 1`). If you select `Pos = Tol` with `stepper_init`, the average position of the limit switch is 0. The only resulting inaccuracy is the fluctuation of the limit switch position measurement when the limit switch position is passed over 4 times.

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).

Because `stepper_init` always stops a previously begun stepper motor movement, you could also use `stepper_init` 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 `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 output ports 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 or via the software, see below, deactivate the limit switch. If a stepper motor movement aborts once (for example, 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 retrigged.

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 code of `RTC6_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 chosen parameters, very long or even infinite stepper motor movements can be initiated by `stepper_init`, `stepper_abs`, for example, 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 (for example, `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 need to separately abort it by setting the CLOCK pulse period (for example, 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 (for example, `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 stop. But here, too, the stepper motor's infinite movement needs to be separately stopped with `stepper_control(Period = 0)`.

WaitTime Parameter

The `WaitTime` parameter of the control commands can be used to set them to return to the calling program after the specified time (in seconds) has elapsed, regardless of whether the stepper motor movement is complete or not.

With `WaitTime = 0`, the command returns immediately. In this case, users must ensure that no unallowed commands are called. In particular, initialization should not be cancelled before it is complete. Otherwise, this leads to incorrect reference positions.

9.1.6 RS-232 Interface

The RS232 socket connector provides an RS-232 interface, see [Chapter 4.6.5 "RS232 Socket Connector", page 82](#).

For configuring the RS-232 interface, see [Chapter 4.6.5 "RS232 Socket Connector", page 82](#).

`rs232_write_data` can be used to send single data words (bytes) to the RS-232 interface. Texts can be sent to the interface by `rs232_write_text` or `rs232_write_text_list`.

9.1.7 McBSP Interface

At the [McBSP interface](#), see [Chapter 4.6.6 "McBSP/ANALOG Socket Connector", page 83](#), a 32-bit data word every 10 µs at DX0 pin (07) is permanently outputted.

The `set_mcbsp_out` and `set_mcbsp_out_ptr` commands allow selection of the signal types (analogously to `set_trigger`) to be outputted there:

- `set_mcbsp_out` lets you specify two signal types for simultaneous output at the [McBSP 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](#)
- `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 interface](#). For a detailed description, see [set_mcbsp_out_ptr](#).

Notes

- For signals and operating conditions, see [Chapter 4.6.6 "McBSP/ANALOG Socket Connector", page 83](#).
- In the default setting, the [McBSP interface](#) always outputs Bit #4...Bit #19 of the Cartesian control values for the x axis and y axis (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 are outputted (with `set_mcbsp_out_ptr` sequentially) until you call one of these two commands again.
- [OIE Output Mode, page 687](#) can be set at the [McBSP interface](#) by `set_mcbsp_out_oie_ctrl` and `set_mcbsp_out_oie_list`.

9.2 Signal Input

Signals of peripherals (for example, 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 Port

The EXTENSION 1 socket connector provides a 16-bit digital TTL input (DIGITAL IN0...15), see [Section "16-Bit Digital Input Port and 16-Bit Digital Output Port", page 77](#), for receiving 16-bit digital values. For synchronization of data transmission, the EXTENSION 1 socket connector outputs 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 [Chapter 9.3.2 "Conditional Command Execution", page 294](#).

9.2.2 2-Bit Digital Input Port

For querying 2-bit digital values, the [LASER Connector](#) provides a 2-bit digital input port (DIGITAL IN1 and DIGITAL IN2), see [Section "2-Bit Digital Input Port", page 73](#).

The input port can be read by `get_laser_pin_in`.

Further commands are provided for conditional command execution (see [Chapter 9.3.2 "Conditional Command Execution", page 294](#)).

9.2.3 RS-232 Interface

The RS232 socket connector provides an RS-232 interface for reading signals, see [Chapter 4.6.5 "RS232 Socket Connector", page 82](#).

For configuring the RS-232 interface, see [Chapter 4.6.5 "RS232 Socket Connector", page 82](#).

Data can be read in by `rs232_read_data`.

9.2.4 McBSP Interface

At the [McBSP interface](#), see [Chapter 4.6.6 "McBSP/ANALOG Socket Connector", page 83](#), the 32-bit data word most recently fully transmitted to the specified memory location can be queried with `read_mcbsp`.

The interpretation as one 32 bit data word or two 16 bit data words is the responsibility of the user.

Signals (position values) received by the [McBSP interface](#) can also be integrated directly into Processing-on-the-fly correction of workpiece or scan-system motion (see [Chapter 8.6 "Processing-on-the-fly", page 241](#) and [Chapter 9.3.4 "Synchronization and Online Positioning by McBSP Signals", page 299](#)) or can be used for an [Online Positioning](#), see [Chapter 8.3.1 "Local Online Positioning", page 227](#) and [Chapter 8.3.2 "Global Online Positioning", page 230](#).

Notes

- For signals and operating conditions, see [Chapter 4.6.6 "McBSP/ANALOG Socket Connector", page 83](#).

9.3 Control by External Signals

The previously described input and output of peripheral signals can be synchronized with scan system control and laser control, as follows:

- The related list commands can be inserted in command lists at appropriate locations.
- Execution of related 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 [Chapter 6.4.2 "List Status", page 106](#) and the list execution status by [get_status](#), see [Chapter 6.4.3 "List Execution Status", page 107](#).
- In addition, the **BUSY** list execution status [List Execution Status](#) is provided by the **BUSY OUT** signal:
 - at the [LASER Connector](#),
see [Section "BUSY List Execution Status", page 73](#)
 - at the [EXTENSION 1](#) socket connector,
see [Section "BUSY List Execution Status", page 78](#)
 - at the [MARKING ON THE FLY](#) socket connector,
see [Section "BUSY List Execution Status", page 81](#)

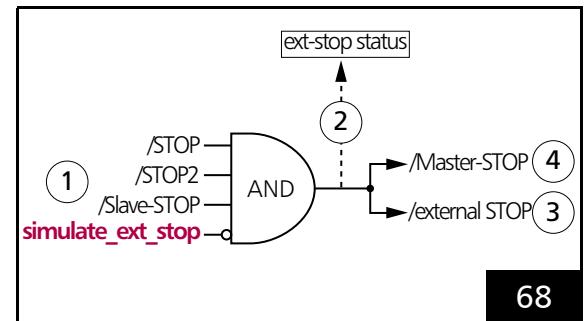
Moreover, the RTC6 PCIe Board provides commands and interfaces (described in the following sections) that allow external control signals (for example, from a light-barrier or from an encoder) to directly control and synchronize execution of command lists or individual commands (including the output of peripheral signals).

Moreover, the RTC6 PCIe Board provides interfaces to control and synchronize list execution directly with external signals.

9.3.1 Starting and Stopping Lists by External Control Signals and Master/Slave Synchronization

External Stop

By a signal at the input ports **/STOP**, **/STOP2** or **/Slave STOP**, or by [simulate_ext_stop](#), an **External Stop** can be initiated, see (1) and (3) in [Figure 68](#).



68

External Stop. See text for description.

This:

- immediately cancels the currently executed list
- switches off the [Signals for "Laser Active" Operation](#) (but does not deactivate them)

Like after calling [stop_execution](#) (internal stop), the following output ports are then set to the previously defined (by [set_port_default](#)) stop-case values given these have not been defined as ≠ “-1”:

- 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
- the 2-bit digital output port
- the two 12-bit analog output ports ([ANALOG OUT1](#) and [ANALOG OUT2](#)) of the [LASER Connector](#)

The input ports for external stop signals (1) are always unlocked so that an external stop can occur at any time (emergency stop).

The /STOP input port is accessible by the **LASER Connector**, see [Section "External Control Signals", page 73](#), the /STOP2 input port at the MARKING ON THE FLY socket connector, see [Section "External Control Signals", page 81](#). Both signal input ports are internally connected to +3.3 V by pull-up resistors (4.7 kΩ). Both input ports are TTL active-LOW and level sensitive. A list abort is triggered as soon as at least one of the two input ports is set to LOW (that is, to 0 V or ground) for at least 10 µs.

If an RTC6 PCIe Board is connected to other boards by the Master or Slave connector, see [Figure 7](#), then external stops from the board receiving the signal are passed on to *all* boards connected to it given the respective master/slave interface has been configured (by **master_slave_config**) accordingly. See also [Chapter 6.6.3 "Master/Slave Operation", page 123](#).

Stops triggered by **stop_execution** are *not* passed on. In contrast, external stops triggered by **simulate_ext_stop** are passed through.

By **get_startstop_info** the current stop status (that is, whether one of the input ports is currently set to LOW) (2) can be queried and whether or not a new **External Stop** has occurred since the last query.

External Start

By a signal at the input ports /START, /START2 or /Slave-START, or by **simulate_ext_start** or **simulate_ext_start_ctrl**, an **External Start** can be initiated (see (1), (5) and (7) in [Figure 69](#)).

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** list execution status (as during list execution) nor the

INTERNAL-BUSY list execution status (as for example, with **goto_xy**) nor the **PAUSED** list execution status (after **pause_list**, **stop_list** or **set_wait**) is set at the moment.

Before the /START, /START2 or /Slave-START input ports (1) can be used, they must be enabled by **set_control_mode** (3).

The control command **simulate_ext_start_ctrl** can be deactivated by **set_control_mode** (Bit #4 = 1). The list command **simulate_ext_start** still remains active.

The /START input port is accessible by the **LASER Connector**, see [Section "External Control Signals", page 73](#), the /START2 input port at the MARKING ON THE FLY socket connector, see [Section "External Control Signals", page 81](#). Both signal input ports are internally connected to +3.3 V by pull-up resistors (4.7 kΩ). Both input ports are TTL active-LOW and edge sensitive (HIGH to LOW level transition). A start is triggered – after activation by **set_control_mode** – as soon as one of the three input signals changes from HIGH to LOW (that is, to 0 V or ground).

If an RTC6 PCIe Board is connected to other boards by the Master or Slave connector, see [Figure 7](#), then external starts from the board receiving the signal are passed on to *all* boards connected to it given the respective master/slave interface has been configured (by **master_slave_config**) accordingly. See also [Chapter 6.6.3 "Master/Slave Operation", page 123](#).

Internal starts triggered by **execute_list** or **execute_at_pointer** are not passed on.

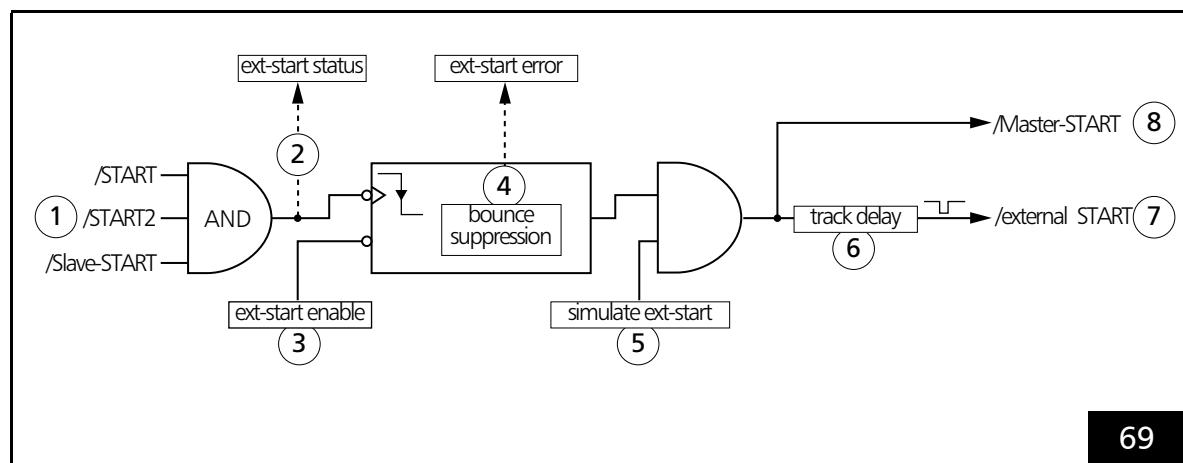
External Starts triggered by **simulate_ext_start** or **simulate_ext_start_ctrl** are passed on, see also Chapter 6.6.3 "Master/Slave Operation", page 123.

get_startstop_info queries the current start status (that is, whether one of the input ports is set to LOW) (2) and whether a list has successfully started since the last query.

bounce_supp enables debouncing of start signals received at the /START, /START2 or /Slave-START input ports (4). Start signals occurring within the defined debouncing time after a successful start signal are thereby suppressed. **get_marking_info** queries whether a start signal has been suppressed (4).

simulate_ext_start_ctrl can be used to start by control command a synchronous start of master/slave-synchronized boards.

The list command **simulate_ext_start** can be used to trigger further starts at defined intervals after the successful one-time External Start (see below).



External Start. See text for description.

External Start with Track Delay

For many applications (for example, if a workpiece must be initially transported from the light barrier to the scan system), the start must be delayed with reference to the triggering start signal.

For this purpose, `set_ext_start_delay`, `set_ext_start_delay_list` or `simulate_ext_start` allow configuring a track delay (see (6) in [Figure 69](#)) that postpones execution of a 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 [Chapter 9.3.3 "Synchronization by Encoder Signals", page 297](#).

External 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`).

By `set_control_mode` (Bit #1 = 1) it can be set that the queue with the external start entries get explicitly cancelled upon an **External Stop**. With `set_control_mode` (Bit #1 = 0) the queue remains existing after an **External Stop**.

Notes

- The /START, /START2 and /Slave-START input ports are *edge* sensitive (HIGH to LOW level *transition*).
- The /STOP, /STOP2 and /Slave-STOP input ports are *level* sensitive.
- A `stop_execution` call disables the /START, /START2 and /Slave-START input ports. An external stop signal also (at least temporarily) disables these input ports, that is, as long as one of the input ports /STOP, /STOP2 or /Slave-STOP is LOW. `set_control_mode` can be used to define whether or not the /START, /START2 and /Slave-START input ports also stay disabled when the external stop signal is no longer active.
- `set_control_mode` additionally allows activation or deactivation of the input ports /START, /START2 or /Slave-START and deactivation of track delay.
- **External Starts** are also suppressed after `pause_list`, `stop_list` or `set_wait` (`PAUSED` list execution status is set). `restart_list`, `stop_execution`, `release_wait` or an **External Stop** ends suppression of the start.
- If list input ports are not yet finished, a buffer flush should be initiated before an **External Start**, for example, by `set_input_pointer` (`get_input_pointer()`), so that any still buffered list commands are fully transferred to list memory, see [Chapter 6.4.1 "Loading Lists", page 104](#).
- If a master board is started internally (for example, 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 has been 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 Starts

By `set_control_mode` and `set_control_mode_list` (Bit #10), equidistant **External 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 Start** – exactly synchronized to the 10 μ s clock of the RTC6 PCIe Board.

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 Start** is *not* counted only beginning with the time point of the triggering external start signal or `simulate_ext_start` (Bit #10 = 0) but already beginning with the most recently executed **External Start** (also executed by an external start signal or `simulate_ext_start`), see **Figure 70**. This makes the distance between consecutive **External Starts** (in encoder pulses) constant.

For activation of this mode, an **External Start** must have successfully occurred (only one-time) in mode Bit #10=0 (Mode &~Bit #10). Each subsequent **External 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
```

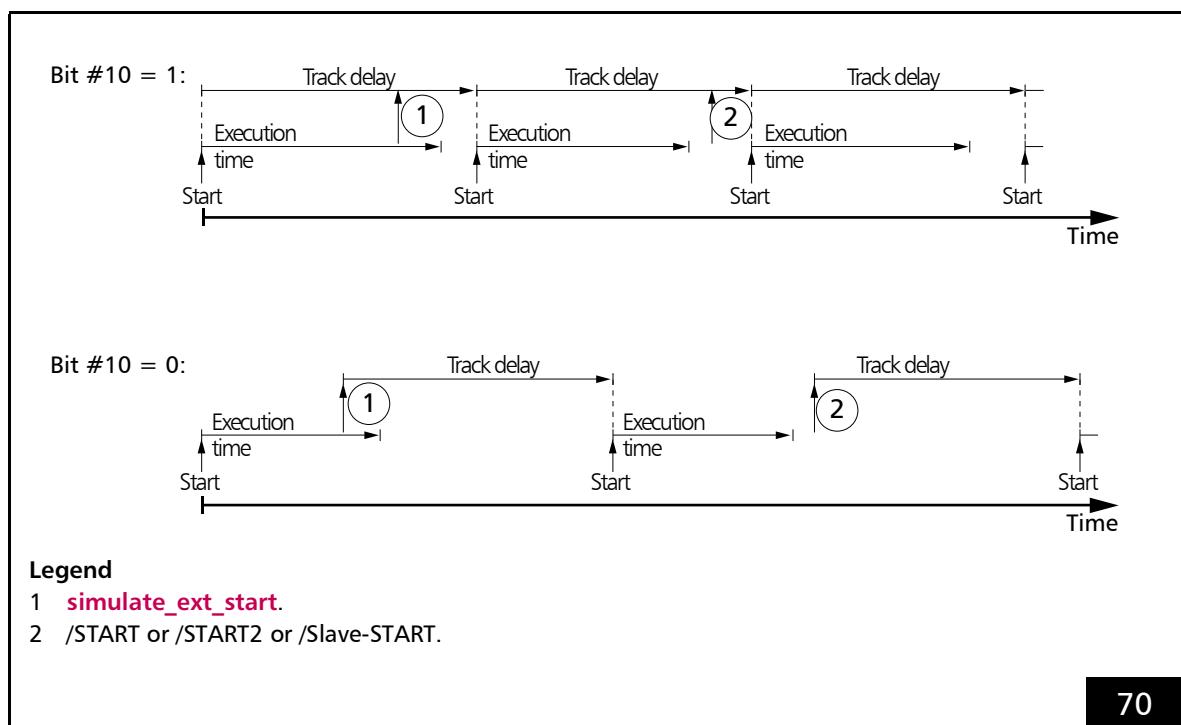
If the first start is to be triggered externally (for example, by /START or by `simulate_ext_start_ctrl`) rather than by an `execute_list_pos` command, but all subsequent starts triggered by `simulate_ext_start`, then `set_control_mode_list` in the above example must be called before `simulate_ext_start`.

After setting `set_control_mode(Bit #1 = 1)`, the external start queue entries get explicitly cancelled upon an **External Stop** (thereby, **External Starts** can be permanently stopped by an **External Stop**).

For `set_control_mode(Bit #1 = 0)` (default setting) – after an otherwise infinitely repetitive series has been stopped (for example, by `set_control_mode(Bit #0 = 0)` – you should deactivate the track delay and cancel the queue of not-yet-executed **External 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` calls.

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.



Regular and irregular **External Starts** (see text for description).

9.3.2 Conditional Command Execution

The so-called conditional commands allow the execution of individual list commands to be made dependent on external control signals.

The conditional commands read out the current value at the 16-bit digital input port at the EXTENSION 1 socket connector, see [Section "16-Bit Digital Input Port and 16-Bit Digital Output Port", page 77](#), and their execution depend on the read out value:

- Conditional Jumps
`list_jump_pos_cond` (synonymous with `list_jump_cond`) and `list_jump_rel_cond` result in either a jump within a buffer area or no jump. The thereby specified jump addresses must fulfill the same conditions as with `list_jump_pos` and `list_jump_rel`
- Variable-distance jump
`switch_ioport` executes a relative list jump
- Conditional Calls of Non-Indexed Subroutines
`list_call_cond` and `list_call_abs_cond` either call or do not call a non-indexed subroutine at a specified memory address
- Conditional Calls of Indexed Subroutines
`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
`set_io_cond_list` and `clear_io_cond_list` associate the output value of the 16-bit digital output port at the EXTENSION 1 socket connector, see [Section "16-Bit Digital Input Port and 16-Bit Digital Output Port", page 77](#), directly with the signals at the digital input port: individual bits of the output port are set or cleared

- Conditional execution of any desired list commands:

`if_cond` and `if_not_cond` have no effect if the condition for the queried value is fulfilled or not. Otherwise, they result in skipping the next list command. In this way, all list commands can be executed conditionally.

Example: the command sequence

`if_cond(...)`

`list_call(...)`

is functionally identical to

`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](#). For this, `if_pin_cond` and `if_not_pin_cond` are available. These 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 μ s.

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 μ s clock cycle. A conditional `pause_list` takes precedence over a simultaneously present /STOP signal. `set_pause_list_not_cond` does the same as `set_pause_list_cond`, if the specified condition is not fulfilled.



Example Code (Pascal)

(1) Confirm a signal:

```
set_start_list(1);
...
// set Bit #0 of the 16-bit digital output port
set_io_cond_list(0, 0, 1);
// loop until the signal is confirmed (that is, Bit #0 of the digital input turns HIGH)
list_jump_rel_cond(0, 1, 0);
// clear Bit #0 of the 16-bit output
clear_io_cond_list(0, 0, 1);
// loop until the signal is confirmed
list_jump_rel_cond(1, 0, 0);
...
set_end_of_list;
execute_list(1);
```

(2) If the lower 4 bits of the digital input have the value (0110), set Bit #1 of the 16-bit digital output.

Otherwise clear Bit #1:

```
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 RTC6 command uses relative addresses and is relocatable:
// skip the next two commands, if the state
// of the 16-bit input port is (xxxx xxxx xxxx 0110)
// list_jump_rel_cond($0006, $0009, 3);

// clear Bit #1 of the 16-bit output port and...
clear_io_cond_list(0, 0, 2);
//RTC4 style: set_list_jump(get_input_pointer + 2);
//this command uses absolute addresses and is not relocatable
//the following RTC6 command uses relative addresses and is relocatable
//...skip the next command
list_jump_rel(2);

// set Bit #1 of the 16-bit output port
set_io_cond_list(0, 0, 2);

// (continue)
...
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
    // call subroutine at address i*100, if [Bit #3..Bit #0] (binary) = i
    list_call_cond(i, 15-i, i*100);
...
```

(4) Choose between 15 indexed subroutines:

```
...
for i := 1 to 15 do
    // call subroutine with index i, if [Bit #3..Bit #0] (binary) = i
    sub_call_cond(i, 15-i, i);
...
```

9.3.3 Synchronization by Encoder Signals

Intended 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 RTC6 can evaluate signals of up to two user-supplied incremental encoders. Though incremental encoders do not 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 RTC6 which can result in incrementing or decrementing of its two internal encoder counters⁽²⁾. The states of the RTC6's encoder counters thereby correspond directly to the position of the workpiece⁽³⁾.

If workpieces are always processed at a constant speed and an encoder is therefore not mandatory, then the encoder signals can also be simulated by **simulate_encoder**, so that the encoder counters are incremented with a constant counting rate of 1 MHz.

The current counts of both encoder counters can be queried by the control command **get_encoder**. Alternatively, they can be stored in a buffer by the list command **store_encoder** and then retrieved from there by the control command **read_encoder**.

In addition, the RTC6 automatically evaluates the current counts if execution of the laser scan processes is controlled as follows:

- For Processing-on-the-fly-applications (see [Chapter 8.6 "Processing-on-the-fly", page 241](#)), the coordinate values of all **Vector commands** and **"Arc" commands** are transformed in accordance with the current encoder counts.
- By the list command **wait_for_encoder_mode**, further execution of a list can be postponed until the selected encoder counter has overstepped or understepped a predefined value.
- With 2D movements, **wait_for_encoder_in_range** waits until the encoder values are within a given rectangle.
- For **External 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 start relative to the triggering input signal or corresponding command, see [Section "External Start", page 289](#).
- Encoder-speed-dependent "Automatic Laser Control", see [Section "Encoder-Speed-Dependent Laser Control", page 203](#), uses the current encoder speed (counter pulses in the most recent 10 µs interval) of an encoder counter to control a laser control signal parameter.

(1) The actual workpiece position can also be forwarded by the **McBSP interface** to the RTC6 PCIe Board (see [Chapter 9.3.4 "Synchronization and Online Positioning by McBSP Signals", page 299](#)).

(2) See [Notice!](#), page 56.

(3) The encoder counters are signed 32-bit counters. 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.

Input Ports for External Encoder Signals

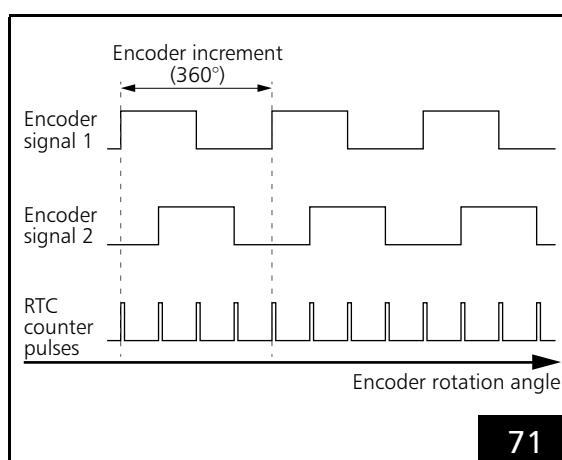
For receiving encoder signals, the MARKING ON THE FLY socket connector provides two encoder input ports (**ENCODER X** and **ENCODER Y**), see [Section "Encoder Input Ports", page 81](#).

For linear movements of the parts to be processed, up to two user-supplied incremental encoders (that define, independently from each other, the workpiece motion in the x direction and y direction) can be connected to these input ports.

For rotational movements only *one* incremental encoder is necessary. For Processing-on-the-fly applications, it must be connected to input port **ENCODER X**.

ENCODER X and **ENCODER Y** are each designed for a pair of standardized differential input signals (RS-422; HIGH level ≥ 2.0 V, LOW level ≤ 0.8 V).

The timing diagram of a typical encoder signal pair shows [Figure 71](#). The second encoder signal is usually phase-shifted by 90° relative to the first signal. The corresponding RTC6-internal encoder counter⁽¹⁾ is triggered at each edge of both signals, that is, one encoder increment results in 4 counter pulses (counts). The relative 90° phase-shift of the two signals allows the RTC6 PCIe Board to detect the movement direction of the workpiece as well. Depending on the direction of movement, the counter value is increased or decreased.



71

Timing diagram of a typical encoder signal pair and of the corresponding RTC5 counter pulses.

The signals at the encoder input port:

- **ENCODER X** trigger encoder counter "Encoder0"
- **ENCODER Y** trigger encoder counter "Encoder1"

The encoder signals should not exceed the maximum allowed frequency of 4 MHz (16 encoder signal edges / μ s).

Encoder Simulation

The encoder simulation can be activated (deactivated) by **simulate_encoder**. The RTC6 PCIe Board provides a 1 MHz clock signal (counter pulses), which replaces the signals of an external incremental encoder.

(1) See [Notice!](#), page 56.

9.3.4 Synchronization and Online Positioning by McBSP Signals

Instead of incremental encoders, the movement between the workpiece and the scan system can be synchronized using absolute position data via the [McBSP interface](#).

Alternatively, this interface also allows the alignment of the workpiece relative to the (stationary) scan system ("Online Positioning").

The input value most recently fully transmitted at the [McBSP interface](#) can be queried by [read_mcbsp](#). In addition, the RTC6 PCIe Board automatically evaluates the current input value if execution of the laser scan processes is controlled as follows:

- For Processing-on-the-fly-applications (see [Chapter 8.6 "Processing-on-the-fly", page 241](#)), the coordinate values of all [Vector command](#) and ["Arc" commands](#) are transformed in accordance with the current input value.
- By the list command [wait_for_mcbsp](#), further execution of a list can be postponed until the input value has reached, overstepped or understepped a predefined value.
- By [Online Positioning](#) (see [Chapter 8.3.1 ""Local Online Positioning""](#), page 227 and [Chapter 8.3.2 ""Global Online Positioning""](#), page 230), 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 interface](#). Compensation of rotational motion, on the other hand, requires transmission of angle positions. During evaluation, full revolutions are suppressed as long as the input values do not exceed the allowed value range (20 full circles, see [set_mcbsp_rot](#)).
- The signals at the [McBSP interface](#) have no impact on the track delay, which can be defined for [External Starts](#) (by [simulate_ext_start](#), [set_ext_start_delay](#) or [set_ext_start_delay_list](#)).

9.4 Periodical I/O Signals

By **periodic_toggle** and **periodic_toggle_list**, periodically repeated signals can be outputted at a selectable IO port:

- ANALOG OUT1 output port, see [Section "12-Bit Analog Output Port 1 and 2", page 73](#)
- ANALOG OUT2 output port, see [Section "12-Bit Analog Output Port 1 and 2", page 73](#)
- 8-bit digital output port, see [Section "8-Bit Digital Output Port", page 80](#)
- 16-bit digital output port, see [Section "16-Bit Digital Input Port and 16-Bit Digital Output Port", page 77](#)
- 2-bit digital output port, see [Section "2-Bit Digital Output Port", page 73](#)

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**.
- **periodic_toggle** and **periodic_toggle_list** toggle endless with **Count** = $2^{32}-1$.

10 RTC6 Commands

10.1 Overview

The following pages describe the complete RTC6 command set (control commands and list commands). The commands are listed according to their intended use. The page numbers refer to [Chapter 10.2 "RTC6 Command Set", page 314](#), where the RTC6 commands (alphabetically ordered) are explained in detail.

10.1.1 Designations

Multi-Board Commands

All commands marked (**n**_) in the following list also exist in a version for multiple RTC6 PCIe Boards installed in one computer. See [Chapter 6.6 "Using Several RTC6 PCIe Boards in One PC", page 122](#) for detailed information about these *multi-board* commands.

An associated multi-board command is also provided for almost all commands. Exceptions are explicitly noted in the description list (in [Chapter 10.2 "RTC6 Command Set", page 314](#)).

Normal, Short, Variable and Multiple List Commands

The list commands of the RTC6 command set vary somewhat in their length of command execution. To differentiate, list commands in the list description (in [Chapter 10.2 "RTC6 Command Set", page 314](#)) are designated as "normal", "short", "variable" and "multi".

- Normal list commands require a full 10 μ s clock cycle 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 μ s clock cycle. In contrast, a short list command that immediately follows a normal list command executes in the subsequent 10 μ s clock cycle.

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 commands** (see **set_laser_pulses**), all without interrupting the **Polyline** (the laser remains on).

In contrast, if a specific time behavior is desired (10 μ s clock cycle), 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 μ s clock cycle. 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). Up to 8 short list commands per 10 μ s clock cycle are possible. However, the maximum number can be lower, depending on the workload of the RTC6 board and the **DSP** mode. Short list commands that alter the output pointer (for example, **sub_call**, **list_return** or **list_jump_pos**) count as two commands. If the maximum number is exceeded, a 10 μ s clock cycle is inserted (equivalent to an additionally inserted **list_continue**, during the **Polyline** the laser remains on).

A maximum of two short list commands per 10 μ s clock cycle 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 μ s clock cycle.

The maximum number of up to 8 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 multiple list commands, see [Section "Multiple List Commands", page 303](#).

Notes

- 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, mark speed, and delays etc.

Undelayed Short List Commands, Delayed Short List Commands

Most short list commands (for example, **list_jump_pos**, **list_call**, **sub_call** or **list_return**) execute before the next list command and prior to a possible scanner delay (**Jump Delay**, **Mark Delay**, **Polygon Delay**). These commands are called “undelayed short list commands” in the corresponding command descriptions (in [Chapter 10.2 “RTC6 Command Set”, page 314](#)).

However, some short list commands only execute after the respective scanner delay (that is, 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 (for example, **set_rot_center_list**, **set_wobbel_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 (for example, **set_trigger** or **save_and_restart_timer**).

Without this delayed execution, such commands would, for example, result in a laser power change already being effective at the end of a **Mark command** (that is, during a still-active **Mark Delay** or **Polygon Delay**) and not at the beginning of the following **Mark command**.

If such a power changing command, such as a peripheral output or **set_laser_power**, is called immediately before a **set_end_of_list**, it is not executed because **set_end_of_list** “clears up” the laser control. Add a **list_nop** to be sure that the command is actually executed.

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.

If these commands are directly before a **set_end_of_list**, add a **list_nop** so that they are still executed within the list.

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.



With several “delayed short list commands” in a sequence of short list commands, only the last “delayed” command can actually be executed delayed. All others before that are executed immediately (yet before a scanner delay).

When you sequence the commands, make sure to place the most important “delayed” command at the end, especially within a [Polyline](#).

Alternatively, you can also explicitly initiate processing of the scanner delay (for example, 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 the normal list command is executed delayed by a 10 μ s clock cycle.

Multiple List Commands

A multiple list command has multiple components that accordingly occupy multiple list storage 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 RTC6 command set currently only contains two-component multiple list commands, for example, [wait_for_encoder_in_range](#) or [set_pixel_line_3d](#).

10.1.2 Compatibility

For RTC6 customers who previously used the RTC4 (RTC5), the command descriptions (in [Chapter 10.2 “RTC6 Command Set”, page 314](#)) note in each command’s “RTC4→RTC6” (“RTC5→RTC6”) section:

- to what extent a command differs from that of the RTC4 (RTC5),
- whether the command is new to the command set,
- whether and how parameter values are converted in [RTC4 Compatibility Mode](#) ([RTC5 Compatibility Mode](#)).

Some RTC3/RTC4 commands and a few RTC2 commands emulated by the RTC3/RTC4 are not supported by the RTC6, see [Chapter 10.3 “Unsupported RTC4/RTC5 Commands”, page 846](#).

Further information on changing from the RTC4 board to RTC6 board see [RTC4 Compatibility Mode](#) and [Chapter 2.10 “Notes for RTC4 Users”, page 47](#).

Further information on changing from the RTC5 board to RTC6 board see [RTC5 Compatibility Mode](#) and [Chapter 2.11 “Notes for RTC5 Users”, page 57](#).



10.1.3 Version Information

Descriptions for a number of commands include version information, listed under “Version info”:

- For newly added commands: the software version in which they become available
- For some older commands: information on implemented changes.

New commands as well as all changes to existing commands are described in

[RTC6_Software_RevisionHistory_<Date>_<Rev>.pdf](#).

10.1.4 Optional Functions

If an option is not present on the RTC6 PCIe Board, see [Chapter 2.6 “Options”, page 37](#), then the associated commands have only partial or non-existent effect:

- Without [Option Processing-on-the-fly](#), commands to activate a Processing-on-the-fly correction have no effect
- Without [Option “3D”](#), 3D vector commands are executed, however, no z axis signals are outputted
- Without [Option “Second Scan Head Control”](#), commands for which the scan head connector can be explicitly specified have not effect on the second scan head connector (for example, [set_matrix](#))

If applicable, the command descriptions contain corresponding notes.

10.1.5 Control Commands

Initialization of the RTC6 DLL

free_RTC6_dll	389
get_RTC_mode	416
init_RTC6_dll	455
set_RTC4_mode	712
set_RTC6_mode	714

Using Several RTC6 PCIe Boards in One PC

acquire_RTC	315
release_RTC	582
rtc6_count_cards	589
select_RTC	598

List Memory Commands

(n_) config_list	343
(n_) get_config_list	393
(n_) get_list_space	410
(n_) load_disk	489
(n_) save_disk	591

Board Initialization and Image Field Correction

(n_) get_head_para	401
(n_) get_sync_status	426
(n_) get_table_para	428
(n_) load_correction_file	485
(n_) load_program_file	500
(n_) load_stretch_table ⁽¹⁾	503
(n_) load_z_table ⁽¹⁾	509
(n_) load_z_table_20b ⁽¹⁾	510
(n_) load_z_table_no ⁽¹⁾	511
(n_) load_z_table_no_20b ⁽¹⁾	512
(n_) number_of_correction_tables	542
read_abc_from_file	568
read_abc_from_file_20b	569
(n_) select_cor_table	594
(n_) set_dsp_mode	621
(n_) sync_slaves	789
write_abc_to_file	834
write_abc_to_file_20b	835

Laser Mode and Parameters

(n_) config_laser_signals	341
(n_) get_standby	419
(n_) load_auto_laser_control	482
(n_) load_position_control	498
(n_) set_auto_laser_control	604
(n_) set_auto_laser_params	608
(n_) set_encoder_speed_ctrl	625
(n_) set_firstpulse_killer	631
(n_) set_laser_control	659
(n_) set_laser_mode	663
(n_) set_laser_pulse_sync	666
(n_) set_laser_pulses_ctrl	668
(n_) set_pulse_picking	709
(n_) set_pulse_picking_length	709
(n_) set_qswitch_delay	710
(n_) set_softstart_level	727
(n_) set_softstart_mode	728
(n_) set_standby	729
(n_) spot_distance_ctrl	767

Setting the Scanner Parameters

(n_) load_varpolydelay	507
(n_) set_delay_mode	619
(n_) set_jump_speed_ctrl	657
(n_) set_mark_speed_ctrl	672
(n_) set_sky_writing	719
(n_) set_sky_writing_limit	720
(n_) set_sky_writing_mode	722
(n_) set_sky_writing_para	724

Coordinate Transformations

(n_) set_angle	602
(n_) set_defocus ⁽¹⁾	615
(n_) set_defocus_offset ⁽¹⁾	617
(n_) set_matrix	673
(n_) set_offset	697
(n_) set_offset_xyz ⁽²⁾	698
(n_) set_scale	715

Online Positioning

(n_) apply_mcbsp	325
(n_) set_mcbsp_global_matrix	677
(n_) set_mcbsp_global_rot	678
(n_) set_mcbsp_global_x	679
(n_) set_mcbsp_global_y	680
(n_) set_mcbsp_matrix	684
(n_) set_mcbsp_rot	690
(n_) set_mcbsp_x	691
(n_) set_mcbsp_y	692

(1) Only with Option "3D".

(2) Limited functionality if no Option "3D".

Status Monitoring and Diagnostics	
(n_) <code>get_head_status</code>	402
(n_) <code>get_overrun</code>	415
(n_) <code>get_value</code>	434
(n_) <code>get_values</code>	436
(n_) <code>get_waveform</code>	438
(n_) <code>get_waveform_offset</code>	439
(n_) <code>measurement_status</code>	537
(n_) <code>stop_trigger</code>	781
iDRIVE Commands	
(n_) <code>control_command</code>	345
(n_) <code>get_transform</code>	431
(n_) <code>read_user_data</code>	580
(n_) <code>send_user_data</code>	601
<code>transform</code>	813
(n_) <code>upload_transform</code>	817
Pixel Output Mode	
(n_) <code>set_default_pixel</code>	614
I/O Commands	
(n_) <code>get_free_variable</code>	398
(n_) <code>get_io_status</code>	406
(n_) <code>get_mcbsp</code>	414
(n_) <code>mcbsp_init</code>	535
(n_) <code>mcbsp_init_spi</code>	536
(n_) <code>periodic_toggle</code>	563
(n_) <code>read_analog_in</code>	570
(n_) <code>read_io_port</code>	573
(n_) <code>read_io_port_buffer</code>	574
(n_) <code>read_mcbsp</code>	576
(n_) <code>read_multi_mcbsp</code>	577
(n_) <code>rs232_config</code>	585
(n_) <code>rs232_read_data</code>	586
(n_) <code>rs232_write_data</code>	587
(n_) <code>rs232_write_text</code>	587
(n_) <code>set_free_variable</code>	649
(n_) <code>set_laser_off_default</code>	663
(n_) <code>set_mcbsp_freq</code>	676
(n_) <code>set_mcbsp_out_ptr</code>	688
(n_) <code>set_port_default</code>	707
(n_) <code>uart_config</code>	816
(n_) <code>write_8bit_port</code>	833
(n_) <code>write_da_1</code>	836
(n_) <code>write_da_2</code>	837
(n_) <code>write_da_x</code>	838
(n_) <code>write_io_port</code>	842
(n_) <code>write_io_port_mask</code>	843
Starting and Stopping Lists by External Control Signals and Master/Slave Synchronization	
(n_) <code>get_counts</code>	393
(n_) <code>get_master_slave</code>	413
(n_) <code>get_startstop_info</code>	420
(n_) <code>set_control_mode</code>	611
(n_) <code>set_extstartpos</code>	628
(n_) <code>set_max_counts</code>	676
(n_) <code>simulate_ext_start_ctrl</code>	765
(n_) <code>sync_slaves</code>	789
List Handling and List Status	
(n_) <code>auto_change</code>	334
(n_) <code>auto_change_pos</code>	335
(n_) <code>get_lap_time</code>	407
(n_) <code>get_status</code>	422
(n_) <code>get_wait_status</code>	437
(n_) <code>pause_list</code>	562
(n_) <code>quit_loop</code>	565
(n_) <code>read_status</code>	578
(n_) <code>release_wait</code>	583
(n_) <code>restart_list</code>	585
(n_) <code>set_pause_list_cond</code>	700
(n_) <code>set_pause_list_not_cond</code>	701
(n_) <code>start_loop</code>	768
(n_) <code>stop_execution</code>	780
(n_) <code>stop_list</code>	780
Input Pointer Commands	
(n_) <code>get_input_pointer</code>	406
(n_) <code>get_list_pointer</code>	409
(n_) <code>load_list</code>	496
(n_) <code>set_input_pointer</code>	651
(n_) <code>set_start_list</code>	731
(n_) <code>set_start_list_1</code>	731
(n_) <code>set_start_list_2</code>	731
(n_) <code>set_start_list_pos</code>	732
Output Pointer Commands	
(n_) <code>execute_at_pointer</code>	381
(n_) <code>execute_list</code>	382
(n_) <code>execute_list_1</code>	382
(n_) <code>execute_list_2</code>	382
(n_) <code>execute_list_pos</code>	383
(n_) <code>get_out_pointer</code>	414

Subroutine Commands		"Classic" Processing-on-the-fly Control Commands	
(n_) <code>copy_dst_src</code>	347	(n_) <code>clear_fly_overflow_ctrl</code>	339
(n_) <code>get_char_pointer</code>	392	(n_) <code>get_encoder</code>	394
(n_) <code>get_sub_pointer</code>	425	(n_) <code>get_fly_2d_offset</code>	398
(n_) <code>get_text_table_pointer</code>	429	(n_) <code>get_marking_info</code>	411
(n_) <code>load_char</code>	484	(n_) <code>init_fly_2d</code>	454
(n_) <code>load_sub</code>	505	(n_) <code>load_fly_2d_table</code>	492
(n_) <code>load_text_table</code>	506	(n_) <code>read_encoder</code>	571
(n_) <code>set_char_pointer</code>	609	(n_) <code>set_ext_start_delay</code>	629
(n_) <code>set_char_table</code>	610	(n_) <code>set_fly_tracking_error</code>	643
(n_) <code>set_sub_pointer</code>	733	(n_) <code>set_mcbsp_in</code> ⁽²⁾	681
(n_) <code>set_text_table_pointer</code>	734	(n_) <code>set_multi_mcbsp_in</code> ⁽²⁾	693
Direct Laser and Scan Head Control		(n_) <code>set_rot_center</code>	711
(n_) <code>disable_laser</code>	349	(n_) <code>simulate_encoder</code>	763
(n_) <code>enable_laser</code>	350	(n_) <code>simulate_ext_stop</code>	766
(n_) <code>get_laser_pin_in</code>	408	Controlling Stepper Motors	
(n_) <code>get_z_distance</code> ⁽¹⁾	440	(n_) <code>get stepper_status</code>	424
(n_) <code>goto_xy</code>	441	(n_) <code>stepper_abs</code>	769
(n_) <code>goto_xyz</code> ⁽¹⁾	442	(n_) <code>stepper_abs_no</code>	770
(n_) <code>laser_signal_off</code>	464	(n_) <code>stepper_control</code>	772
(n_) <code>laser_signal_on</code>	465	(n_) <code>stepper_disable_switch</code>	773
(n_) <code>set_laser_pin_out</code>	664	(n_) <code>stepper_enable</code>	774
Version Commands		(n_) <code>stepper_init</code>	775
(n_) <code>get_bios_version</code>	391	(n_) <code>stepper_rel</code>	777
<code>get_dll_version</code>	394	(n_) <code>stepper_rel_no</code>	778
(n_) <code>get_hex_version</code>	404	Jump Mode	
(n_) <code>get_RTC_version</code>	417	(n_) <code>get_jump_table</code>	407
(n_) <code>get_serial_number</code>	418	(n_) <code>load_jump_table</code>	493
Error Commands		(n_) <code>load_jump_table_offset</code>	494
(n_) <code>get_error</code>	395	(n_) <code>set_jump_mode</code>	653
(n_) <code>get_last_error</code>	408	(n_) <code>set_jump_table</code>	658
(n_) <code>reset_error</code>	584	Control Commands only for Scan Systems with SCAnahead Control	
(n_) <code>set_verify</code>	747	(n_) <code>activate_scanahead_autodelays</code>	324
<code>verify_checksum</code>	819	(n_) <code>get_scanahead_params</code>	417
Date, Time, Serial Numbers		(n_) <code>set_scanahead_laser_shifts</code>	716
(n_) <code>get_list_serial</code>	410	(n_) <code>set_scanahead_line_params</code>	716
(n_) <code>get_serial</code>	418	(n_) <code>set_scanahead_params</code>	716
(n_) <code>select_serial_set</code>	600	(n_) <code>set_scanahead_speed_control</code>	716
(n_) <code>set_serial</code>	717		
(n_) <code>set_serial_step</code>	718		
(n_) <code>time_update</code>	794		

(1) Limited functionality if no Option "3D".

(2) Limited functionality if no Option Processing-on-the-fly.

Control Commands for RTC6 Ethernet Boards

<code>(n_) eth_assign_card</code>	351
<code>eth_assign_card_ip</code>	352
<code>(n_) eth_check_connection</code>	354
<code>(n_) eth_configure_link_loss</code>	355
<code>eth_convert_ip_to_string</code>	356
<code>eth_convert_string_to_ip</code>	357
<code>eth_count_cards</code>	358
<code>eth_found_cards</code>	359
<code>eth_get_card_info</code>	360
<code>eth_get_card_info_search</code>	361
<code>(n_) eth_get_com_timeouts</code>	362
<code>(n_) eth_get_com_timeouts_auto</code>	363
<code>(n_) eth_get_error</code>	364
<code>eth_get_ip</code>	365
<code>eth_get_ip_search</code>	365
<code>(n_) eth_get_last_error</code>	366
<code>(n_) eth_get_port_numbers</code>	368
<code>eth_get_serial_search</code>	368
<code>(n_) eth_get_standalone_status</code>	369
<code>(n_) eth_get_static_ip</code>	370
<code>eth_max_card</code>	371
<code>eth_remove_card</code>	372
<code>eth_search_cards</code>	373
<code>eth_search_cards_range</code>	374
<code>(n_) eth_set_com_timeouts</code>	375
<code>(n_) eth_set_com_timeouts_auto</code>	376
<code>(n_) eth_set_high_performance_mode</code>	377
<code>(n_) eth_set_port_numbers</code>	378
<code>eth_set_search_cards_timeout</code>	379
<code>(n_) eth_set_static_ip</code>	380
<code>(n_) time_control_eth</code>	791

Standalone Functionality for RTC6 Ethernet Boards

<code>(n_) eth_boot_dcmd</code>	353
<code>(n_) eth_boot_timeout</code>	353
<code>(n_) set_eth_boot_control</code>	627
<code>(n_) read_image_eth</code>	572
<code>(n_) store_program</code>	782
<code>(n_) write_image_eth</code>	841

Other Control Commands

<code>(n_) auto_cal</code>	331
<code>(n_) bounce_supp</code>	336
<code>(n_) create_dat_file</code>	348
<code>(n_) get_auto_cal</code>	390
<code>(n_) get_card_type</code>	391
<code>(n_) get_galvo_controls</code>	399
<code>(n_) get_hi_data</code>	404
<code>(n_) get_hi_pos</code>	405
<code>(n_) get_temperature</code>	428
<code>(n_) get_time</code>	429
<code>(n_) get_timestamp_long</code>	430
<code>(n_) home_position</code>	443
<code>(n_) home_position_xyz</code> ⁽¹⁾	444
<code>(n_) load_zoom_correction_file</code>	513
<code>(n_) move_to</code>	541
<code>(n_) set_hi</code>	650
<code>(n_) set_mcbsp_out_oie_ctrl</code>	687
<code>(n_) set_pause_list_cond</code>	700
<code>(n_) set_timelag_compensation</code>	735
<code>(n_) set_zoom</code>	762
<code>(n_) store_timestamp_counter</code>	783
<code>(n_) write_hi_pos</code>	840

(1) Limited functionality if no Option "3D".

10.1.6 List Commands

Meanings:

nor	normal list command
var	variable list command
us	undelayed short list command
ds	delayed short list command
mul	multiple list command

Board Initialization and Image Field Correction

(n_) `select_cor_table_list` var 597

2D Jump Commands

(n_) <code>jump_abs</code> nor	457
(n_) <code>jump_rel</code> nor	459
(n_) <code>para_jump_abs</code> nor	543
(n_) <code>para_jump_rel</code> nor	545
(n_) <code>timed_jump_abs</code> nor	797
(n_) <code>timed_jump_rel</code> nor	799
(n_) <code>timed_para_jump_abs</code> nor	805
(n_) <code>timed_para_jump_rel</code> nor	807

3D Jump Commands⁽¹⁾

(n_) <code>jump_abs_3d</code> nor	458
(n_) <code>jump_rel_3d</code> nor	460
(n_) <code>para_jump_abs_3d</code> nor	544
(n_) <code>para_jump_rel_3d</code> nor	546
(n_) <code>timed_jump_abs_3d</code> nor	798
(n_) <code>timed_jump_rel_3d</code> nor	800
(n_) <code>timed_para_jump_abs_3d</code> mul	806
(n_) <code>timed_para_jump_rel_3d</code> mul	808

micro_vector[*] Commands

(n_) <code>micro_vector_abs</code> nor	538
(n_) <code>micro_vector_abs_3d</code> nor	539
(n_) <code>micro_vector_rel</code> nor	540
(n_) <code>micro_vector_rel_3d</code> nor	541

2D Mark Commands

(n_) <code>arc_abs</code> nor	327
(n_) <code>arc_rel</code> nor	329
(n_) <code>mark_abs</code> nor	515
(n_) <code>mark_ellipse_abs</code> nor	522
(n_) <code>mark_ellipse_rel</code> nor	523
(n_) <code>mark_rel</code> nor	524
(n_) <code>para_mark_abs</code> nor	549
(n_) <code>para_mark_rel</code> nor	552
(n_) <code>set_ellipse</code> us	623
(n_) <code>timed_arc_abs</code> nor	795
(n_) <code>timed_arc_rel</code> nor	796
(n_) <code>timed_mark_abs</code> nor	801
(n_) <code>timed_mark_rel</code> nor	803
(n_) <code>timed_para_mark_abs</code> nor	809
(n_) <code>timed_para_mark_rel</code> nor	811

3D Mark Commands⁽¹⁾

(n_) <code>arc_abs_3d</code> nor	328
(n_) <code>arc_rel_3d</code> nor	330
(n_) <code>mark_abs_3d</code> nor	516
(n_) <code>mark_rel_3d</code> nor	525
(n_) <code>para_mark_abs_3d</code> nor	551
(n_) <code>para_mark_rel_3d</code> nor	553
(n_) <code>timed_mark_abs_3d</code> nor	802
(n_) <code>timed_mark_rel_3d</code> nor	804
(n_) <code>timed_para_mark_abs_3d</code> mul	810
(n_) <code>timed_para_mark_rel_3d</code> mul	812

Text Commands

(n_) <code>mark_char</code> us	517
(n_) <code>mark_char_abs</code> us	518
(n_) <code>mark_text</code> var	529
(n_) <code>mark_text_abs</code> var	530
(n_) <code>select_char_set</code> us	593

Date, Time, Serial Numbers

(n_) <code>mark_date</code> nor	519
(n_) <code>mark_date_abs</code> nor	521
(n_) <code>mark_serial</code> nor	526
(n_) <code>mark_serial_abs</code> nor	528
(n_) <code>mark_time</code> nor	531
(n_) <code>mark_time_abs</code> nor	533
(n_) <code>select_serial_set_list</code> us	600
(n_) <code>set_serial_step_list</code> nor	718
(n_) <code>time_fix</code> nor	792
(n_) <code>time_fix_f</code> nor	792
(n_) <code>time_fix_f_off</code> nor	793

(1) Limited functionality if no **Option "3D"**.

Status Monitoring and Diagnostics		Setting the Scanner Parameters	
(n_) <code>set_trigger</code> ^{ds}	736	(n_) <code>set_delay_mode_list</code> ^{mul}	620
(n_) <code>set_trigger4</code> ^{ds}	744	(n_) <code>set_jump_speed</code> ^{us}	657
Starting and Stopping Lists by External Control Signals and Master/Slave Synchronization		(n_) <code>set_mark_speed</code> ^{ds}	672
(n_) <code>set_control_mode_list</code> ^{nor}	613	(n_) <code>set_scanner_delays</code> ^{ds}	717
(n_) <code>set_extstartpos_list</code> ^{us}	628	(n_) <code>set_sky_writing_limit_list</code> ^{us}	720
List Handling and Structured Programming		(n_) <code>set_sky_writing_list</code> ^{nor}	721
(n_) <code>list_continue</code> ^{nor}	472	(n_) <code>set_sky_writing_mode_list</code> ^{nor}	723
(n_) <code>list_jump_pos</code> ^{us}	474	(n_) <code>set_sky_writing_para_list</code> ^{nor}	726
(n_) <code>list_jump_rel</code> ^{us}	476		
(n_) <code>list_next</code> ^{us}	478	Coordinate Transformations	
(n_) <code>list_nop</code> ^{nor}	478	(n_) <code>set_angle_list</code> ^{var}	603
(n_) <code>list_repeat</code> ^{us}	479	(n_) <code>set_defocus_list</code> ^{var (1)}	616
(n_) <code>list_until</code> ^{us}	481	(n_) <code>set_defocus_offset_list</code> ^{var (1)}	618
(n_) <code>long_delay</code> ^{nor}	514	(n_) <code>set_matrix_list</code> ^{var}	675
(n_) <code>set_end_of_list</code> ^{nor}	626	(n_) <code>set_offset_list</code> ^{var}	697
(n_) <code>set_list_jump</code> ^{us}	671	(n_) <code>set_offset_xyz_list</code> ^{var (2)}	699
(n_) <code>set_wait</code> ^{nor}	748	(n_) <code>set_scale_list</code> ^{var}	715
Subroutine Commands		Online Positioning	
(n_) <code>list_call</code> ^{us}	466	(n_) <code>apply_mcbsp_list</code> ^{nor}	326
(n_) <code>list_call_abs</code> ^{us}	468	(n_) <code>set_mcbsp_global_matrix_list</code> ^{us}	677
(n_) <code>list_call_abs_repeat</code> ^{us}	469	(n_) <code>set_mcbsp_global_rot_list</code> ^{us}	678
(n_) <code>list_call_repeat</code> ^{us}	471	(n_) <code>set_mcbsp_global_x_list</code> ^{us}	679
(n_) <code>list_return</code> ^{us}	480	(n_) <code>set_mcbsp_global_y_list</code> ^{us}	680
(n_) <code>sub_call</code> ^{us}	784	(n_) <code>set_mcbsp_matrix_list</code> ^{us}	685
(n_) <code>sub_call_abs</code> ^{us}	785	(n_) <code>set_mcbsp_rot_list</code> ^{us}	690
(n_) <code>sub_call_abs_repeat</code> ^{us}	786	(n_) <code>set_mcbsp_x_list</code> ^{us}	691
(n_) <code>sub_call_repeat</code> ^{us}	787	(n_) <code>set_mcbsp_y_list</code> ^{us}	692
Setting the Laser Parameters		Direct Laser and Scan Head Control	
(n_) <code>config_laser_signals_list</code> ^{ds}	342	(n_) <code>laser_on_list</code> ^{var}	461
(n_) <code>set_auto_laser_params_list</code> ^{ds}	608	(n_) <code>laser_on_pulses_list</code> ^{var}	462
(n_) <code>set_encoder_speed</code> ^{ds}	624	(n_) <code>laser_signal_off_list</code> ^{nor}	464
(n_) <code>set_firstpulse_killer_list</code> ^{us}	631	(n_) <code>laser_signal_on_list</code> ^{nor}	465
(n_) <code>set_laser_delays</code> ^{us}	662	(n_) <code>para_laser_on_pulses_list</code> ^{var}	547
(n_) <code>set_laser_pin_out_list</code> ^{us}	664		
(n_) <code>set_laser_pulses</code> ^{ds}	667	Pixel Output Mode	
(n_) <code>set_laser_timing</code> ^{ds}	669	(n_) <code>set_default_pixel_list</code> ^{us}	614
(n_) <code>set_pulse_picking_list</code> ^{us}	710	(n_) <code>set_n_pixel</code> ^{var}	696
(n_) <code>set_qswitch_delay_list</code> ^{us}	710	(n_) <code>set_pixel</code> ^{var}	702
(n_) <code>set_softstart_level_list</code> ^{nor}	727	(n_) <code>set_pixel_line</code> ^{nor}	703
(n_) <code>set_softstart_mode_list</code> ^{var}	728	(n_) <code>set_pixel_line_3d</code> ^{mul (2)}	706
(n_) <code>set_standby_list</code> ^{ds}	730		
(n_) <code>set_vector_control</code> ^{us}	745		
(n_) <code>spot_distance</code> ^{us}	767		

(1) Only with Option "3D".

(2) Limited functionality if no Option "3D".

I/O Commands

(n_)	clear_io_cond_list	us	340
(n_)	periodic_toggle_list	us	564
(n_)	read_io_port_list	us	575
(n_)	rs232_write_text_list	var	588
(n_)	set_free_variable_list	us	649
(n_)	set_io_cond_list	us	652
(n_)	set_laser_power	us	665
(n_)	set_mcbsp_out	us	686
(n_)	set_mcbsp_out_ptr_list	mul	689
(n_)	set_port_default_list	us	708
(n_)	write_8bit_port_list	ds	833
(n_)	write_da_1_list	ds	836
(n_)	write_da_2_list	ds	837
(n_)	write_da_x_list	ds	839
(n_)	write_io_port_list	ds	842
(n_)	write_io_port_mask_list	ds	843
(n_)	write_port_list	us	844

Conditional Commands

(n_)	if_cond	us	445
(n_)	if_not_cond	us	449
(n_)	if_not_pin_cond	us	452
(n_)	if_pin_cond	us	453
(n_)	list_call_abs_cond	us	469
(n_)	list_call_cond	us	470
(n_)	list_jump_cond	us	473
(n_)	list_jump_pos_cond	us	475
(n_)	list_jump_rel_cond	us	477
(n_)	sub_call_abs_cond	us	785
(n_)	sub_call_cond	us	786
(n_)	switch_iport	us	788

"Classic" Processing-on-the-fly

List Commands

(n_)	activate_fly_2d	var (1)	320
(n_)	activate_fly_2d_encoder	mul (1)	321
(n_)	activate_fly_xy	var (1)	323
(n_)	activate_fly_xy_encoder	mul (1)	323
(n_)	clear_fly_overflow	us	339
(n_)	fly_return	nor	384
(n_)	fly_return_z	nor	388
(n_)	if_fly_x_overflow	us	445
(n_)	if_fly_y_overflow	us	446
(n_)	if_fly_z_overflow	us	447
(n_)	if_not_activated	us	448
(n_)	if_not_fly_x_overflow	us	450
(n_)	if_not_fly_y_overflow	us	451
(n_)	if_not_fly_z_overflow	us	452
(n_)	park_position	var (1)	554
(n_)	park_return	var (1)	558
(n_)	set_ext_start_delay_list	nor	630
(n_)	set_fly_2d	nor (1)	635
(n_)	set_fly_limits	us	639
(n_)	set_fly_limits_z	us	640
(n_)	set_fly_rot	nor (1)	641
(n_)	set_fly_rot_pos	nor (1)	642
(n_)	set_fly_x	nor (1)	644
(n_)	set_fly_x_pos	nor (1)	645
(n_)	set_fly_y	nor (1)	646
(n_)	set_fly_y_pos	nor (1)	647
(n_)	set_fly_z	nor (1)	648
(n_)	set_mcbsp_in_list	nor (2)	683
(n_)	set_multi_mcbsp_in_list	nor (2)	695
(n_)	set_rot_center_list	ds	711
(n_)	simulate_ext_start	nor	764
(n_)	store_encoder	us	781
(n_)	wait_for_encoder	nor	824
(n_)	wait_for_encoder_in_range	mul	825
(n_)	wait_for_encoder_in_range_mode	mul	826
(n_)	wait_for_encoder_mode	nor	827
(n_)	wait_for_mcbsp	nor	829

(1) Only with Option Processing-on-the-fly.

(2) Limited functionality if no Option Processing-on-the-fly.

"Fly Extension" List Commands

(n_) <code>activate_fly_1_axis</code> var	317
(n_) <code>activate_fly_2_axes</code> var	318
(n_) <code>fly_return_1_axis</code> var	385
(n_) <code>fly_return_2_axes</code> var	386
(n_) <code>fly_return_3_axes</code> var	387
(n_) <code>park_position_1_axis</code> var	556
(n_) <code>park_position_2_axes</code> var	557
(n_) <code>park_return_1_axis</code> var	560
(n_) <code>park_return_2_axes</code> var	561
(n_) <code>set_fly_1_axis</code> nor	632
(n_) <code>set_fly_2_axes</code> nor	633
(n_) <code>set_fly_3_axes</code> mul	637
(n_) <code>wait_for_1_axis</code> nor	820
(n_) <code>wait_for_2_axes</code> mul	822

Controlling Stepper Motors

(n_) <code>stepper_abs_list</code> us	770
(n_) <code>stepper_abs_no_list</code> us	771
(n_) <code>stepper_control_list</code> us	772
(n_) <code>stepper_enable_list</code> us	774
(n_) <code>stepper_rel_list</code> us	777
(n_) <code>stepper_rel_no_list</code> us	778
(n_) <code>stepper_wait</code> nor	779

Jump Mode

(n_) <code>set_jump_mode_list</code> nor	656
--	-----

Camming

(n_) <code>camming</code> nor	337
-------------------------------------	-----

Wobbel Mode

(n_) <code>set_wobbel</code> ds	749
(n_) <code>set_wobbel_control</code> us	751
(n_) <code>set_wobbel_direction</code> us	753
(n_) <code>set_wobbel_mode</code> ds	754
(n_) <code>set_wobbel_mode_phase</code> ds	756
(n_) <code>set_wobbel_offset</code> ds	757
(n_) <code>set_wobbel_vector</code> us	758
(n_) <code>set_wobbel_vector_2</code> us	762

List Commands only for Scan Systems with SCANAhead Control

(n_) <code>activate_scanahead_autodelays_list</code> us ..	324
(n_) <code>set_scanahead_laser_shifts_list</code> us ..	716
(n_) <code>set_scanahead_line_params_list</code> us ..	716

List Commands for a Restricted User Group only⁽¹⁾

(n_) <code>regulation3</code> nor	581
(n_) <code>set_duty_cycle_table</code> nor	622
(n_) <code>set_laser_timing_table</code> nor	670

Other List Commands

(n_) <code>jump_abs_drill</code> nor	458
(n_) <code>jump_abs_drill_2</code> nor	458
(n_) <code>jump_rel_drill</code> nor	460
(n_) <code>jump_rel_drill_2</code> nor	460
(n_) <code>range_checking</code> us	566
(n_) <code>save_and_restart_timer</code> ds	590
(n_) <code>set_mcbsp_out_oie_list</code> us	762
(n_) <code>set_zoom_list</code> us	762
(n_) <code>store_timestamp_counter_list</code> us	783
(n_) <code>wait_for_timestamp_counter</code> nor	830
(n_) <code>wait_for_timestamp_counter_long</code> nor ..	831
(n_) <code>wait_for_timestamp_counter_mode</code> nor ..	832

(1) These users have been informed separately in how to handle these RTC6 commands. Therefore, only the command table is available in this manual and further information is omitted.

10.1.7 Data Types

The following table defines the formats and ranges of the different data types used by the RTC6 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 value		double	double	double
pointer to a \0-terminated ANSI string (1 byte per char)	4 Byte for Win32-user programs 8 Byte for Win64-user programs	pchar	char*	string

Pointer to Locations in the PC Memory

Some RTC6 commands (for example, `get_transform`, `get_values`, `get_waveform`, `transform` or `upload_transform`) have pointers to locations in the PC 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-user programs,
`ULONG_PTR = unsigned 64-bit value`
 for Win64-user programs):

```
#if !defined(ULONG_PTR)
#define ULONG_PTR ULONG
#endif // !defined(ULONG_PTR)
```

Usually, the data type `ULONG_PTR` is also appropriately defined in the Windows header file `BaseTsd.h`.



10.2 RTC6 Command Set

The commands are in alphabetical order.

The general structure of the command tables is as follows⁽¹⁾:

- (1) A program language-neutral form is used. Each real programming language has its own individual naming.

Category of the command	<code>example_command_name_one</code>				
Function	Short description describing the purpose of the command.				
Call	Shows the correct spellings and the sequence of the parameters. Note, there is no semicolon at the end of the line. A '&' (address operator) is only used in this table row and indicates a pointer. Examples: <code>example_command_name_one(parameter_A, &parameter_B, parameter_C)</code> <code>example_variable = example_command_name_one(parameter_A)</code>				
Parameters(*)	<table border="0"> <tr> <td>A</td> <td>Short text. Data type. (*) in some C/C++ code descriptions labeled with <code>_out</code>.</td> </tr> <tr> <td>C</td> <td>Short text. Data type.</td> </tr> </table>	A	Short text. Data type. (*) in some C/C++ code descriptions labeled with <code>_out</code> .	C	Short text. Data type.
A	Short text. Data type. (*) in some C/C++ code descriptions labeled with <code>_out</code> .				
C	Short text. Data type.				
Returned parameter values(**)	<table border="0"> <tr> <td>B</td> <td>Short text. Data type. (**) in some C/C++ code descriptions labeled with <code>_out</code>.</td> </tr> </table>	B	Short text. Data type. (**) in some C/C++ code descriptions labeled with <code>_out</code> .		
B	Short text. Data type. (**) in some C/C++ code descriptions labeled with <code>_out</code> .				
Result	If implemented: mentions the returned result value and data type in generic form (Example: error code #. As an unsigned 32-bit value.). A value range is here only given, if the actual usable one is smaller than the value range of the data type. If not implemented: "None."				
Comments	<ul style="list-style-type: none"> • Additional information on this command. • References to other chapters and publications. 				
RTC4→RTC6	States the differences to the command (of the same name) of the RTC4 command set.				
RTC5→RTC6	States the differences to the command (of the same name) of the RTC5 command set.				
Version info	For example, states the minimum versions of DLL, RBF, OUT which are required to use the command, latest change in version.				
References	Links to related commands: <code>command_name_two</code> , <code>command_name_three</code>				

Ctrl Command	acquire_rtc
Function	Acquires the specified RTC6 board for a user program.
Call	<code>NoOfAcquiredCard = acquire_rtc(CardNo)</code>
Parameters	CardNo RTC6 DLL -internal number (RTC6 board management index) of the desired board. As an unsigned 32-bit value.
Result	<p>The return value is:</p> <ul style="list-style-type: none"> • CardNo, if the acquisition has been successful • 0, if the board is currently acquired by another user program or the version check detects an error <p>As an unsigned 32-bit value.</p>
Comments	<ul style="list-style-type: none"> • acquire_rtc is also useful for single-board systems which need to coordinate the use of a RTC6 board by different user programs. • acquire_rtc has no effect (return value 0, get_last_error return code RTC6_PARAM_ERROR), if: <ul style="list-style-type: none"> – CardNo > number of RTC6 PCIe Boardns found during initialization (see rtc6_count_cards) and no RTC6 Ethernet Board is entered there – CardNo = 0 (real boards begin with 1) • Access rights to existing boards are granted exclusively (always to only one user program at a time). Therefore, acquire_rtc has no effect if the specified board is already acquired by another user program (return value 0, get_last_error return code RTC6_ACCESS_DENIED). This must explicitly release the RTC6 board (by release_rtc or free_rtc6_dll) before the RTC6 board can be acquired by another user program (with acquire_rtc). On the other hand, if the board has been already freed prior to initialization of a user program, then initialization by init_rtc6_dll result in RTC6 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 CardNo. • Assorted versions of the RTC6 DLL and the files RTC6OUT.out, RTC6RBF.rbf and RTC6DAT.dat cannot be arbitrarily combined with another. acquire_rtc performs a version compatibility check. If RTC6OUT.out, RTC6RBF.rbf and RTC6DAT.dat are not yet loaded, then this check cannot be explicitly executed (get_last_error return code RTC6_TIMEOUT), but the check still regarded as successful and acquisition is not hindered. If RTC6OUT.out, RTC6RBF.rbf and RTC6DAT.dat are loaded and the version check determines an error, then access is denied (return value 0, get_last_error return code RTC6_ACCESS_DENIED RTC6_VERSION_MISMATCH). With RTC6 Ethernet Boards, RTC6OUT.out must be replaced by RTC6ETH.out. init_rtc6_dll does not automatically acquire RTC6 Ethernet Boards. • With RTC6 Ethernet Boards, acquire_rtc can take up to 1 s. • A board successfully acquired by acquire_rtc does not automatically become the "active" board. Activation of a board is only achieved by select_rtc or init_rtc6_dll. • Running boards are neither halted nor initialized by acquire_rtc.



Ctrl Command	acquire_RTC
Comments (cont'd)	<ul style="list-style-type: none">• acquire_RTC is available even without explicit access rights to a particular RTC6 board.• acquire_RTC is not available as a multi-board command.• See also Chapter 6.7.1 "Notes on Board Acquisition by a User Program", page 127.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	init_RTC6_dll , select_RTC , free_RTC6_dll , release_RTC , RTC6_count_cards

Variable List Command	activate_fly_1_axis
Function	"Fly Extension" Command: Activates a 1 Axisn-Processing-on-the-fly application.
Restriction	If the Option Processing-on-the-fly is not enabled, then activate_fly_1_axis terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	activate_fly_1_axis(Axis, Mode, Scale, Offset)
Parameters	<p>Axis Axis from Table 3, page 258. As an unsigned 32-bit value. Allowed values: 1...2.</p> <p>Mode Mode from Table 4, page 260. As an unsigned 32-bit value. Allowed values: 1...4.</p> <p>Scale Scaling factor. As a 64-bit IEEE floating point value. Allowed value range: • $1/256 \leq \text{Scale} \leq 16.000,0$ with linear axis (1 or 2) Scale can be + or -. Only the absolute value is restricted.</p> <p>Offset Offset to the encoder value. As a signed 32-bit value.</p>
Comments	<ul style="list-style-type: none"> Being an "Fly Extension" Command, activate_fly_1_axis must not be used mixed with "Classic" Processing-on-the-fly commands (see Footnote, page 241). See Chapter 8.6 "Processing-on-the-fly", page 241 and Section ""Fly Extension" Commands", page 258. activate_fly_1_axis occupies two list memory positions. activate_fly_1_axis requires two $10 \mu\text{s}$ clock cycles for execution. At the specified Axis1, neither a Processing-on-the-fly correction nor a rotation correction must be active. However, the Axis can be combined as a linear axis with the third linear axis. An xy positioning stage compensation is automatically executed, if after the successful activate_fly_1_axis execution Axis 1 and Axis 2 are activated with two physically different encoder modes (for example, Mode 1 and 3 mean the same physical encoder) and xy positioning stage compensation is defined and activated (see load_fly_2d_table), then xy travel table compensation is automatically executed. With an unallowed parameter value, activate_fly_1_axis is replaced by a list_nop (get_last_error return code RTC6_PARAM_ERROR). See also comments on activate_fly_2_axes.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 617, OUT 617, RBF 623.
References	activate_fly_2_axes

Variable List Command	activate_fly_2_axes
Function	"Fly Extension" Command: Activates a 2-Axes-Processing-on-the-fly application.
Restriction	If the Option Processing-on-the-fly is not enabled, then activate_fly_2_axes terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	<code>activate_fly_2_axes(ModeX, ScaleX, OffsetX, ModeY, ScaleY, OffsetY)</code>
Parameters	<p>ModeX Mode from Table 4, page 260. As an unsigned 32-bit value.</p> <p>ScaleX Scaling factor. As a 64-bit IEEE floating point value. Allowed value range: • $1/256 \leq \text{Scale} \leq 16.000,0$ with linear axis (1 or 2) Scale can be + or -. Only the absolute value is restricted.</p> <p>OffsetX Offset to the encoder value. As a signed 32-bit value.</p> <p>ModeY Like ModeX.</p> <p>ScaleY Like ScaleX.</p> <p>OffsetY Like OffsetX.</p>
Comments	<ul style="list-style-type: none"> Being an "Fly Extension" Command, activate_fly_2_axes must not be used mixed with "Classic" Processing-on-the-fly commands (see Footnote, page 241). See Chapter 8.6 "Processing-on-the-fly", page 241 and Section "Fly Extension" Commands, page 258. The Axes are automatically 1 and 2. The modes need to be two physically different encoder modes (Mode 1 and 3 or 2 and 4 are not allowed). activate_fly_2_axes occupies two list memory positions. activate_fly_2_axes requires two $10 \mu\text{s}$ clock cycles for execution. At Axis 1 and 2, neither a Processing-on-the-fly correction nor a rotation correction must be active. However, the Axes can be combined as linear axes with the third linear axis. With an unallowed parameter value, activate_fly_2_axes is replaced by a list_nop (get_last_error return code RTC6_PARAM_ERROR). If an xy positioning stage compensation is defined and active (see load_fly_2d_table), then it is carried out automatically. See also comments on activate_fly_2d (encoder reset, error bit, get_marking_info).



Variable List Command	activate_fly_2_axes
Comments (cont'd)	<ul style="list-style-type: none"> The following command calls are executed in the same way: <ul style="list-style-type: none"> – activate_fly_2_axes(1, ScaleX, 0, 2, ScaleY, 0) = <code>activate_fly_2d(ScaleX, ScaleY)</code> or <code>activate_fly_xy(ScaleX, ScaleY)</code> – activate_fly_2_axes(1, ScaleX, EncX, 2, ScaleY, EncY) = <code>activate_fly_xy_encoder(ScaleX, ScaleY, EncX, EncY)</code> “xy” and “2d” are distinguished with <code>wait_for_1_axis</code> and <code>wait_for_2_axes</code>! – activate_fly_2_axes(1, ScaleX, OffsetX, 2, ScaleY, OffsetY) = <code>{</code> <code>activate_fly_1_axis(1, 1, ScaleX, OffsetX);</code> <code>activate_fly_1_axis(2, 2, ScaleY, OffsetY);</code> <code>}</code>
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 617, OUT 617, RBF 623.
References	activate_fly_1_axis

Variable List Command	activate_fly_2d
Function	Activates a set_fly_2d Processing-on-the-fly application without encoder resets.
Restriction	If the Option Processing-on-the-fly is not enabled, then activate_fly_2d terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	<code>activate_fly_2d(ScaleX, ScaleY)</code>
Parameters	<p>ScaleX Scaling factor as for activate_fly_2d_encoder.</p> <p>ScaleY Like ScaleX.</p>
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 8.6.4 "Compensating 2D Motions", page 248. Unlike set_fly_2d, activate_fly_2d does not thereby reset the encoders, but instead calculates 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 29-bit virtual Image field, then Processing-on-the-fly correction is not activated. Then an error bit is set that can be queried by get_marking_info (Bit #9). 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). 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 jump to an appropriate error handling routine. Successful activation by activate_fly_2d does <i>not</i> reset an error bit. It remains set for get_marking_info until get_marking_info is called. If unallowed parameter values are supplied (for example, for ScaleX = 0), then activate_fly_2d is (already during loading) replaced by a list_nop (get_last_error return code RTC6_PARAM_ERROR). activate_fly_2d does not affect the laser control signals (Signals for "Laser Active" Operation remain on/off if they are on/off).
RTC4→RTC6	New command. RTC4 Compatibility Mode: see set_fly_2d .
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_fly_2d, activate_fly_2d_encoder, activate_fly_xy

Multiple List Command	activate_fly_2d_encoder
Function	Activates a set_fly_2d Processing-on-the-fly application with encoder reset and encoder offsets.
Restriction	If the Option Processing-on-the-fly is not enabled, then activate_fly_2d_encoder terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	activate_fly_2d_encoder(ScaleX, ScaleY, EncX, EncY)
Parameters	<p>ScaleX Scaling factor as for set_fly_2d.</p> <p>ScaleY Like ScaleX.</p> <p>EncX Encoder offset. As a signed 32-bit value.</p> <p>EncY Like EncX.</p>
Comments	<ul style="list-style-type: none"> • activate_fly_2d_encoder occupies two list memory positions and also needs two 10 μs clock cycles to execute (the first part of activate_fly_2d_encoder is <i>not</i> a short list command). • activate_fly_2d_encoder is a combination of set_fly_2d (encoder reset) and activate_fly_2d (see also comments there). However, in activate_fly_2d_encoder the current (reset) encoder values are not used to calculate the Processing-on-the-fly-uncorrected virtual Image field coordinates, but the parameter values EncX and EncY. For the error handling, see comments of activate_fly_2d. • Because of this combination, activate_fly_2d_encoder saves the positioning stage movement (which is often long but may be necessary for the Processing-on-the-fly activation without this command) from the initialization position (for example, at the lower left corner) to the center and back again. • Subsequently all encoder values are offset with EncX and EncY, before the Processing-on-the-fly correction is applied. All other encoder related commands refer to the actual encoder values and behave as before. • If the value of EncX or EncY is not allowed (that is, the Processing-on-the-fly-corrected virtual Image field coordinates are outside the virtual Image field limits), then activate_fly_2d_encoder is replaced by a list_nop (get_last_error return code RTC6_PARAM_ERROR). • If the value of ScaleX or ScaleY is not allowed (see set_fly_2d), the first part is transferred to the board (and thus executes the encoder reset). However, the second part is replaced by a list_nop (get_last_error return code RTC6_PARAM_ERROR). • With active Processing-on-the-fly correction, activate_fly_2d_encoder is a short list command without further effect and merely sets an error bit queryable by get_marking_info (Bit #9). Therefore, activate_fly_2d_encoder cannot be used to modify the Processing-on-the-fly mode itself nor the scaling factors or encoder offsets of the same mode.



Multiple List Command	activate_fly_2d_encoder
RTC4→RTC6	New command. RTC4 Compatibility Mode: see set_fly_2d .
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 610, OUT 610, RBF 615.
References	set_fly_2d , activate_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 Option Processing-on-the-fly is not enabled, then activate_fly_xy 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 factor as for set_fly_x . ScaleY Scaling factor as for set_fly_y .
Comments	<ul style="list-style-type: none"> Like activate_fly_2d with the difference that a set_fly_x/set_fly_y Processing-on-the-fly session is activated.
RTC4→RTC6	New command. RTC4 Compatibility Mode: see set_fly_x / set_fly_y .
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_fly_x , set_fly_y , activate_fly_xy_encoder

Multiple List Command	activate_fly_xy_encoder
Function	Activates a set_fly_x / set_fly_y Processing-on-the-fly application with encoder reset and encoder offsets.
Restriction	If the Option Processing-on-the-fly is not enabled, then activate_fly_xy_encoder terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	<code>activate_fly_xy_encoder(ScaleX, ScaleY, EncX, EncY)</code>
Parameters	ScaleX Scaling factor as for set_fly_x . ScaleY Scaling factor as for set_fly_y . EncX Encoder offset. As a signed 32-bit value. EncY Encoder offset. As a signed 32-bit value.
Comments	<ul style="list-style-type: none"> Like activate_fly_2d_encoder with the difference that a set_fly_x/set_fly_y Processing-on-the-fly session is activated.
RTC4→RTC6	New command. RTC4 Compatibility Mode: see set_fly_2d .
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 610, OUT 610, RBF 615.
References	set_fly_x , set_fly_y , activate_fly_xy , activate_fly_2d_encoder



Ctrl Command	activate_scanahead_autodelays
Comments	<ul style="list-style-type: none">Only for scan systems with SCANAhead technology, for example, the excelliSCAN.This command is described in the "excelliSCAN Scan Heads – Functional Principle of SCANAhead Servo Control and Operation by RTC6 Boards" Manual.

Undelayed Short List Command	activate_scanahead_autodelays_list
Comments	<ul style="list-style-type: none">Only for scan systems with SCANAhead technology, for example, the excelliSCAN.This command is described in the "excelliSCAN Scan Heads – Functional Principle of SCANAhead Servo Control and Operation by RTC6 Boards" Manual.

Ctrl Command	apply_mcbsp
Function	Queries the most recent values fully transmitted over the McBSP interface for "Local Online Positioning" and defines offset and/or rotation matrix M_R or general transformation matrix M_T for subsequent coordinate transformations.
Call	<code>apply_mcbsp(HeadNo, 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>first</i> scan head connector. = 2: The definition only affects the <i>second</i> scan head connector. = 0, 3: The definition affects <i>both</i> scan head connectors. Only the two least significant bits are evaluated.</p> <p>at_once Determines when the defined transformation becomes effective. As an unsigned 32-bit value. = 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 currently BUSY list execution status or INTERNAL-BUSY list execution status is set) 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 is executed. > 2: Like <code>at_once = 2</code>.</p>
Comments	<ul style="list-style-type: none"> Data acquisition by the McBSP interface for "Local Online Positioning" must be activated in advance by 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, apply_mcbsp only defines (as with set_offset) an x offset and/or y offset or also (as with set_angle) a rotation matrix or (as with set_matrix) a general matrix operation (see Chapter 8.3.1 ""Local Online Positioning"", page 227). As with the commands described in Chapter 8.2 "Coordinate Transformations", page 223, the parameter at_once determines when the newly defined total transformation becomes effective. Transformations previously defined by set_angle, set_offset or set_matrix 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 interface. The McBSP interface ignores the first FrameSync signal after a load_program_file or mcbsp_init. That is, data provided is not transmitted, see Section "RTC6 PCIe Board as Receiver", page 84.

Ctrl Command	apply_mcbsp
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	apply_mcbsp_list , set_mcbsp_x , set_mcbsp_y , set_mcbsp_rot , set_mcbsp_matrix

Normal List Command	apply_mcbsp_list				
Function	Like apply_mcbsp , but a list command.				
Call	<code>apply_mcbsp_list(HeadNo, at_once)</code>				
Parameters	<table> <tr> <td>HeadNo</td> <td>Like apply_mcbsp.</td> </tr> <tr> <td>at_once</td> <td> <p>Determines when the defined transformation becomes effective. As an unsigned 32-bit value.</p> <p>= 0: The transformation settings are only accumulated and intermediately stored, but the transformation is not processed as long as this is not activated by another coordinate transformation (for example, by a list command with <code>at_once = 1</code> or a corresponding control command).</p> <p>= 1: The transformation is immediately calculated (including all transformation settings that were accumulated until then) and processed prior to the next list command.</p> <p>= 2: The transformation settings are only accumulated and intermediately stored (as with <code>at_once = 0</code>). However, The transformation is immediately calculated (including all transformation settings that were accumulated and intermediately stored until then) and applied to the current position when the next jump_abs or jump_rel (only if no list is currently being executed: also goto_xy or goto_xyz) is executed.</p> <p>> 2: Like <code>at_once = 2</code>.</p> </td> </tr> </table>	HeadNo	Like apply_mcbsp .	at_once	<p>Determines when the defined transformation becomes effective. As an unsigned 32-bit value.</p> <p>= 0: The transformation settings are only accumulated and intermediately stored, but the transformation is not processed as long as this is not activated by another coordinate transformation (for example, by a list command with <code>at_once = 1</code> or a corresponding control command).</p> <p>= 1: The transformation is immediately calculated (including all transformation settings that were accumulated until then) and processed prior to the next list command.</p> <p>= 2: The transformation settings are only accumulated and intermediately stored (as with <code>at_once = 0</code>). However, The transformation is immediately calculated (including all transformation settings that were accumulated and intermediately stored until then) and applied to the current position when the next jump_abs or jump_rel (only if no list is currently being executed: also goto_xy or goto_xyz) is executed.</p> <p>> 2: Like <code>at_once = 2</code>.</p>
HeadNo	Like apply_mcbsp .				
at_once	<p>Determines when the defined transformation becomes effective. As an unsigned 32-bit value.</p> <p>= 0: The transformation settings are only accumulated and intermediately stored, but the transformation is not processed as long as this is not activated by another coordinate transformation (for example, by a list command with <code>at_once = 1</code> or a corresponding control command).</p> <p>= 1: The transformation is immediately calculated (including all transformation settings that were accumulated until then) and processed prior to the next list command.</p> <p>= 2: The transformation settings are only accumulated and intermediately stored (as with <code>at_once = 0</code>). However, The transformation is immediately calculated (including all transformation settings that were accumulated and intermediately stored until then) and applied to the current position when the next jump_abs or jump_rel (only if no list is currently being executed: also goto_xy or goto_xyz) is executed.</p> <p>> 2: Like <code>at_once = 2</code>.</p>				
Comments	<ul style="list-style-type: none"> • See apply_mcbsp. 				
RTC4→RTC6	New command.				
RTC5→RTC6	Unchanged functionality.				
Version info	Available as of DLL 600, OUT 600, RBF 600.				
References	apply_mcbsp				

Normal List Command	arc_abs
Function	Moves the laser focus from the current position at mark speed along an arc with the specified angle and center point (absolute coordinate values) within a 2D Image field .
Call	arc_abs(<i>X</i> , <i>Y</i> , <i>Angle</i>)
Parameters	<p><i>X</i> Absolute x coordinate of the arc center. In bits. As a signed 32-bit value. Allowed value range: [-268,435,456...+268,435,455]. Out-of-range values are clipped to the boundary values.</p> <p><i>Y</i> Like <i>X</i> (analogously).</p> <p><i>Angle</i> Arc angle. In degrees. As a 64-bit IEEE floating point value. A positive sign means "clockwise". Allowed value range: [-3,600.0°...+3,600.0°] (± 10 full circles). Out-of-range values are clipped to the boundary values.</p>
Comments	<ul style="list-style-type: none"> If the mark speed has not been previously explicitly set by set_mark_speed or set_mark_speed_ctrl, then the marking is executed at a predefined mark speed of 1,000 <i>bits/ms</i>. The Signals for "Laser Active" Operation are automatically turned on at the beginning of the marking (or remain on after a directly preceding [*]mark[*] Command or "Arc" command). The defined Scanner Delays and Laser Delays are thereby taken into account, see Chapter 7.2 "Delay Settings – Coordinating Scan Head Control and Laser Control", page 144. Note that other delays are executed in Sky Writing mode. Exception: zero-length "Arc" commands, see Section "Notes", page 148.
RTC4→RTC6	Unchanged functionality. In addition: increased value range. In RTC4 Compatibility Mode , the RTC6 multiplies the specified values for <i>X</i> and <i>Y</i> by 16. The allowed value ranges decrease accordingly.
RTC5→RTC6	Unchanged functionality. In addition: increased value range.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_mark_speed , set_scanner_delays , arc_rel , timed_arc_abs , mark_abs , arc_abs_3d , mark_ellipse_abs

Normal List Command	<code>arc_abs_3d</code>								
Function	Moves the laser focus at mark speed from the current position helical 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 helical axis and the z end coordinate are specifiable as absolute coordinate values.								
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see select_cor_table), then <code>arc_abs_3d</code> has the same effect as arc_abs .								
Call	<code>arc_abs_3d(X, Y, Z, Angle)</code>								
Parameters	<table> <tr> <td>X</td><td>Position of the helical axis (parallel to the z axis) as absolute x coordinate. In bits. As a signed 32-bit value. Allowed value range: [-268,435,456...+268,435,455]. Out-of-range values are clipped to the boundary values.</td></tr> <tr> <td>Y</td><td>Like X (analogously).</td></tr> <tr> <td>Z</td><td>Absolute z end coordinate. In bits. As a signed 32-bit value. Allowed value range: [-524,288...+524,287]. Out-of-range values are clipped to the boundary values.</td></tr> <tr> <td>Angle</td><td>Arc angle. In degrees. As a 64-bit IEEE floating point value. A positive sign means "clockwise". Allowed value range: [-3,600.0°...+3,600.0°] (± 10 full circles). Out-of-range values are clipped to the boundary values.</td></tr> </table>	X	Position of the helical axis (parallel to the z axis) as absolute x coordinate. In bits. As a signed 32-bit value. Allowed value range: [-268,435,456...+268,435,455]. Out-of-range values are clipped to the boundary values.	Y	Like X (analogously).	Z	Absolute z end coordinate. In bits. As a signed 32-bit value. Allowed value range: [-524,288...+524,287]. Out-of-range values are clipped to the boundary values.	Angle	Arc angle. In degrees. As a 64-bit IEEE floating point value. A positive sign means "clockwise". Allowed value range: [-3,600.0°...+3,600.0°] (± 10 full circles). Out-of-range values are clipped to the boundary values.
X	Position of the helical axis (parallel to the z axis) as absolute x coordinate. In bits. As a signed 32-bit value. Allowed value range: [-268,435,456...+268,435,455]. Out-of-range values are clipped to the boundary values.								
Y	Like X (analogously).								
Z	Absolute z end coordinate. In bits. As a signed 32-bit value. Allowed value range: [-524,288...+524,287]. Out-of-range values are clipped to the boundary values.								
Angle	Arc angle. In degrees. As a 64-bit IEEE floating point value. A positive sign means "clockwise". Allowed value range: [-3,600.0°...+3,600.0°] (± 10 full circles). Out-of-range values are clipped to the boundary values.								
Comments	<ul style="list-style-type: none"> Except for the additional motion in the third dimension, <code>arc_abs_3d</code> functions similarly to arc_abs (see comments there). The z motion is not taken into account during calculation of the number of Microsteps. 								
RTC4→RTC6	<p>New command.</p> <p>In RTC4 Compatibility Mode, the RTC6 multiplies the values specified for X, Y and Z by 16. The allowed value range decreases accordingly.</p>								
RTC5→RTC6	<p>Unchanged functionality. In addition: increased value range.</p> <p>In RTC5 Compatibility Mode, the RTC6 multiplies the values specified for Z by 16. The allowed value range decreases accordingly.</p>								
Version info	Available as of DLL 600, OUT 600, RBF 600.								
References	arc_abs , arc_rel_3d								

Normal List Command	<code>arc_rel</code>
Function	Moves the laser focus from the current position at mark speed along an arc with the specified angle and center point (relative coordinate values) within a 2D Image field .
Call	<code>arc_rel(dx, dy, Angle)</code>
Parameters	<p><code>dx</code> Relative x coordinate of the arc center. In bits. As a signed 32-bit value. Allowed value range: [-268,435,456...+268,435,455]. Out-of-range values are clipped to the boundary values.</p> <p><code>dy</code> Like <code>dx</code> (analogously).</p> <p><code>Angle</code> Arc angle. In degrees. As a 64-bit IEEE floating point value. A positive sign means "clockwise". Allowed value range: [-3,600.0°...+3,600.0°] (± 10 full circles). Out-of-range values are clipped to the boundary values.</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, <code>arc_rel</code> is identical to <code>arc_abs</code> (see comments there).
RTC4→RTC6	Unchanged functionality. In addition: increased value range. In RTC4 Compatibility Mode , the RTC6 multiplies the specified values for <code>dx</code> and <code>dy</code> by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality. In addition: increased value range.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_mark_speed , set_scanner_delays , arc_abs , timed_arc_rel , mark_rel , arc_rel_3d , mark_ellipse_rel

Normal List Command	<code>arc_rel_3d</code>
Function	Moves the laser focus at mark speed from the current position helical 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 helical axis and the z end coordinate are specifiable as relative coordinate values.
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see select_cor_table), then <code>arc_rel_3d</code> has the same effect as <code>arc_rel</code> .
Call	<code>arc_rel_3d(dx, dy, dz, Angle)</code>
Parameters	<p><code>dx</code> Position of the helical axis (parallel to the z axis) as relative x coordinate. In bits. As a signed 32-bit value. Allowed value range: [-268,435,456...+268,435,455]. Out-of-range values are clipped to the boundary values.</p> <p><code>dy</code> Like <code>dx</code> (analogously).</p> <p><code>dz</code> Relative z end coordinate. In bits. As a signed 32-bit value. Allowed value range: [-524,288...+524,287]. Out-of-range values are clipped to the boundary values.</p> <p><code>Angle</code> Arc angle. In degrees. As a 64-bit IEEE floating point value. A positive sign means "clockwise". Allowed value range: [-3,600.0°...+3,600.0°] (± 10 full circles). Out-of-range values are clipped to the boundary values.</p>
Comments	<ul style="list-style-type: none"> The position of the helical axis (<code>dx</code>, <code>dy</code>) and the z end coordinate (<code>dz</code>) are to be specified as relative coordinates with respect to the current position. Otherwise, <code>arc_rel_3d</code> is identical to <code>arc_abs_3d</code> (see comments there).
RTC4→RTC6	<p>New command.</p> <p>RTC4 Compatibility Mode: see <code>arc_abs_3d</code>.</p>
RTC5→RTC6	Unchanged functionality. In addition: increased value range.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	<code>arc_abs_3d</code> , <code>arc_rel</code>

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: First scan head connector. = 2: Second scan head connector. Requires Option "Second Scan Head Control".</p> <p>Command Control parameter. As an unsigned 32-bit value. Allowed value range: [0...4]. = 0: The RTC6 detects the current Home-In positions, stores them in the RTC6 DLL and in the Flash memory as Home-In reference values and initializes the gain and offset values (Gain = 1.0, Offset = 0). = 1: The RTC6 detects the current Home-In positions, calculates and sets the new gain and offset values and thereby activates drift compensation. = 2: The RTC6 deactivates drift compensation by initializing the gain and offset values (Gain = 1.0, Offset = 0). = 3: The RTC6 detects the current Home-In positions (but – in comparison to <code>Command = 1</code> – leaves the gain and offset values unchanged and that is, does not activate drift compensation) = 4: The RTC6 checks the ASC hardware 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 Flash memory.</p>
Result	<p>Error code or type of sensor system. As an unsigned 32-bit value.</p> <p>3 auto_cal cannot be executed because the BUSY list execution status or INTERNAL-BUSY list execution status is currently set.</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.</p> <p>9, 90 Sensor for x axis or y axis 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 Download error. The values have possibly not been saved. For this error, the get_last_error return code RTC6_FLASH_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 usage of auto_cal, see Chapter 8.10 "Automatic Self-Calibration", page 274. At the end of auto_cal execution, the RTC6 board 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 galvanometer scanners. After first-time or renewed connecting a scan system, a reference value determination should be performed by auto_cal(Command = 0). Otherwise, the RTC6 uses the stored reference values of a previously operated (possibly different) scan system when later performing an automatic self-calibration.

Ctrl Command	auto_cal
Comments (cont'd)	<ul style="list-style-type: none"> After initialization of the RTC6, drift compensation is turned off (gain = 1.0, offset = 0). However, previously determined reference values are still available. 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. <code>auto_cal</code> is not executed, if a <code>HeadNo</code> or <code>Command</code> value is invalid (return value 6, <code>get_last_error</code> return code <code>RTC6_PARAM_ERROR</code>). This also applies for <code>HeadNo</code> = 2 if the Option "Second Scan Head Control" is not enabled (return value 6). <code>auto_cal</code> is not executed (return value 3, <code>get_last_error</code> return code <code>RTC6_BUSY</code>), if: <ul style="list-style-type: none"> the <code>BUSY</code> list execution status is set the <code>INTERNAL-BUSY</code> list execution status is set <code>auto_cal</code> is even executed, if: <ul style="list-style-type: none"> a list has been paused by <code>set_wait</code> (<code>PAUSED</code> list execution status set) For <code>RTC6_PARAM_ERROR</code>, the <code>BUSY</code> list execution status is not checked; therefore return codes <code>RTC6_BUSY</code> and <code>RTC6_PARAM_ERROR</code> do not occur simultaneously. For <code>Command</code> = 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>. 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 <code>Flash memory</code> and can be subsequently queried by <code>get_auto_cal</code>. For automatic <code>Command</code> = 4 execution, however, no corresponding return value is generated. The return value is instead simply that of the primary <code>Command</code> call (see above). When an error occurs, a corresponding error code gets saved to the <code>Flash memory</code> (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 <code>Flash memory</code> by explicitly calling <code>auto_cal</code>(Command = 0) or <code>auto_cal</code>(Command = 4). <code>auto_cal</code>(Command = 1) and <code>auto_cal</code>(Command = 3) cannot be executed as long as the error is still in the <code>Flash memory</code>. The error is <i>not</i> cleared by these calls from the <code>Flash memory</code>.



Ctrl Command	auto_cal
RTC4→RTC6	The functions for automatic self-calibration (<code>Command = 0...2</code>) are (largely) unchanged. New: <code>Command = 3</code> and <code>Command = 4</code> .
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_hi , get_hi_pos , get_auto_cal , write_hi_pos

Ctrl Command	auto_change
Function	Activates a one-time automatic list change.
Call	<code>auto_change()</code>
Comments	<ul style="list-style-type: none"> • <code>auto_change</code> is synonymous with <code>auto_change_pos(0)</code>. This starts the subsequent list at its beginning. • See also comments for auto_change_pos.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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(<i>Pos</i>)
Parameters	<p><i>Pos</i> 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.</p>
Comments	<ul style="list-style-type: none"> • auto_change_pos triggers a subsequent <i>one-time-only</i> list change or a list new start. For further list changes or list new starts, auto_change_pos must be called again. • auto_change_pos can be called at any time. • 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 (<i>Mem2</i> = 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 "Changing Lists Automatically", page 109. • 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 has been already processed and the other list has been started (at the supplied start position). If no list has been 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: <i>Pos</i> \geq <i>Mem1</i> or <i>Pos</i> \geq <i>Mem2</i>), then the start position is set to the beginning of the list (<i>Pos</i> = 0). • If, during processing of a list, the auto_change_pos(<i>Pos</i> > 0) and start_loop are called, then upon the next set_end_of_list the command auto_change_pos(<i>Pos</i> > 0) is executed; and at the next one the start_loop command is executed.

Ctrl Command	auto_change_pos
Comments (cont'd)	<ul style="list-style-type: none"> The current List Status values can be queried by read_status. The current List Execution Status values can be queried by get_status. auto_change_pos triggers a flush of the buffered list input, see Chapter 6.4.1 "Loading Lists", page 104.
RTC4→RTC6	<p>Basically unchanged functionality. However:</p> <p>The list memory address (Pos) is supplied to the RTC6 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 (Mem2 = 0), then the RTC6 command behaves like the RTC4 command.</p>
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	auto_change, get_status, read_status

Ctrl Command	bounce_supp
Function	Debounces the external start signal.
Call	bounce_supp(Length)
Parameters	Length Debouncing time. In ms. As an unsigned 32-bit value. Allowed value range: [0...1023]. The 22 higher bits are ignored.
Comments	<ul style="list-style-type: none"> bounce_supp enables debouncing of start signals received at the /START, /START2 or /Slave-START input ports, see Section "External Start", page 289. 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). Increase the debouncing time until this additional signal is no longer detected. The debouncing time default value is 0 ms.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 values: [0...(2²³-1)].</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 values: [1...(2²³-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: RTC6 board controls laser as with a [*]mark[*] Command. camming terminates automatically. = 1: User controls laser. camming terminates automatically. = 2: User controls laser. camming does <i>not</i> terminate automatically. The Camming command list is executed until the end point (0 or NPos-1) and then waits for a new external /START. = 3: User controls laser. camming does <i>not</i> terminate automatically. The Camming command list is continuously processed in circulation (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 values: $2^{-60} < Scale < 2^{60}$.</p> <p>Code Is not used. As an unsigned 32-bit value.</p>
Comments	<ul style="list-style-type: none"> • See also Chapter 8.11 "Camming", page 278. • If there are unallowed parameter values, camming is replaced by a list_nop already during loading (get_last_error return code RTC6_PARAM_ERROR).

Normal List Command	camming
Comments (cont'd)	<ul style="list-style-type: none"> 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 then the full responsibility of the user. If Ctrl = 0 or 1, then camming terminates automatically (in the next cycle) as soon as the index first undershoots 0 or overshoots NPos-1 (the final Camming command to be executed is then be the one with an index of 0 or NPos-1). For these two modes (as always, if a list is BUSY list execution status), no External Starts are allowed as long as camming has not yet terminated. In contrast, control modes Ctrl = 2 and 3 are endless. Here, the Camming process can only be terminated by stop_execution or an External Stop. However, in these two modes (as an exception) External Starts are allowed if the list (or camming) is still active. If Ctrl = 2, then the index is executed until the end point (0 or NPos-1) and then waits for a new external /START. If Ctrl = 3, then the index is set to modulus NPos (whereby the encoder speed is supposed not to be so high that a complete rotation is skipped). Thus, Ctrl = 3 works like a ring buffer. camming is a normal list command, but with a variable execution period. camming functions even if the Option Processing-on-the-fly is not activated. While camming is executing, get_out_pointer always provides the position of camming, not the position of the current index.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_encoder_speed , set_auto_laser_control

Undelayed Short List Command	clear_fly_overflow
Function	Resets the specified error bits (from get_marking_info) for customer-defined monitoring of Processing-on-the-fly applications.
Call	<code>clear_fly_overflow(Mode)</code>
Parameters	<p>Mode The error bits to be reset. As an unsigned 32-bit value.</p> <p>Bit #0 = 1: error bit Bit #4 (underflow X). Bit #1 = 1: error bit Bit #5 (overflow X). Bit #2 = 1: error bit Bit #6 (underflow Y). Bit #3 = 1: error bit Bit #7 (overflow Y). Bit #4 = 1: error bit Bit #24 (underflow Z). Bit #5 = 1: error bit Bit #25 (overflow Z). Bit #0...Bit #5 can be combined as desired. Higher-order bits are ignored.</p>
Comments	<ul style="list-style-type: none"> For usage of clear_fly_overflow, see Section "Customer-Defined Monitoring Area", page 255. All 6 error bits are reset for: <ul style="list-style-type: none"> Mode = 0 Mode = 63
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	get_marking_info , clear_fly_overflow_ctrl

Ctrl Command	clear_fly_overflow_ctrl
Function	Like clear_fly_overflow , but a control command.
Call	<code>clear_fly_overflow_ctrl(Mode)</code>
Parameters	Mode Like clear_fly_overflow .
Comments	<ul style="list-style-type: none"> See clear_fly_overflow.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 628, OUT 629.
References	clear_fly_overflow

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 input port on the EXTENSION 1 socket connector meets the following condition: $((\text{IOvalue AND Mask1}) = \text{Mask1}) \text{ AND } (((\text{not IOvalue}) \text{ AND Mask0}) = \text{Mask0})$ (= 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	<p><code>Mask1</code> 16-bit mask. As an unsigned 32-bit value. Only the lower 16 bits are evaluated.</p> <p><code>Mask0</code> See <code>Mask1</code>.</p> <p><code>MaskClear</code> See <code>Mask1</code>.</p>
Comments	<ul style="list-style-type: none"> • <code>clear_io_cond_list</code> 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 Port and 16-Bit Digital Output Port", page 77 and Chapter 9.3.2 "Conditional Command Execution", page 294.
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 Bit #1 to Bit #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→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 output signal types to be outputted on pin (01) (LASER1), pin (02) (LASERON) and pin (09) (LASER2) of the LASER Connector .
Call	<code>config_laser_signals(Config)</code>
Parameters	<p>Config Desired signal configuration. As an unsigned 32-bit value.</p> <p>The following bits configure:</p> <ul style="list-style-type: none"> Bit #0...Bit #1: Pin (02) (LASERON channel) Bit #2...Bit #3: Pin (01) (LASER1 channel) Bit #4...Bit #5: Pin (09) (LASER2 channel) Bit #6: Ignored ... Bit #31: Ignored <p>Meanings:</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: <code>Config = 100100_b = 0x24 = 36.</code></p>
Comments	<ul style="list-style-type: none"> • The specified configuration takes effect the next time the laser is switched on. Therefore, config_laser_signals should not be called during an active marking procedure. • See also set_laser_control, Bit #3 and Bit #4.
Examples	<ul style="list-style-type: none"> • <code>Config = 000111_b</code>: 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. • <code>Config = 100100_b</code> (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 RTC6 only generates LASER1 signals if the LASERON signal is on).
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	config_laser_signals_list , set_laser_control



Delayed Short List Command	config_laser_signals_list
Function	Like config_laser_signals , but a list command.
Call	config_laser_signals_list(Config)
Parameters	Config Like config_laser_signals .
Comments	<ul style="list-style-type: none">• See config_laser_signals.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	config_laser_signals

Ctrl Command	config_list
Function	Configures the list memory, that is, assigns specific memory locations to the list memory areas.
Call	<code>config_list(Mem1, Mem2)</code>
Parameters	Mem1 Storage positions for list memory area "List 1". As an unsigned 32-bit value.
	Mem2 Storage positions for list memory area "List 2". As an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> The RTC6 list memory contains 8,388,608 (= 2^{23}) storage positions in total. They can be divided into three areas ("lists") by config_list. The sizes of "List 1" and "List 2" are specified by the parameters Mem1 and Mem2. All remaining storage positions not assigned to "List 1" or "List 2" are automatically assigned by config_list to the protected list memory area "List 3". The following rules apply to the parameters Mem1 and Mem2 (invalid values are automatically corrected in the specified order): <ul style="list-style-type: none"> – Mem1 > 0 ("List 1" must not be empty). Mem1 = 0 is corrected to Mem1 = 1. – Mem1 ≤ 2^{23} ("List 1" can contain a maximum of 2^{23} storage positions). Mem1 > 2^{23} is corrected to Mem1 = 2^{23}. – Mem1 = "-1" is interpreted as Mem1 = $(2^{32}-1)$ and corrected to Mem1 = 2^{23}. Example: With <code>config_list(-1, x)</code> where x is any value (also x = -1), "List 1" is automatically assigned the entire list memory (Mem1 = 2^{23}, Mem2 = 0, no memory for "List 3"). – Mem2 ≤ $2^{23} - \text{Mem1}$ ("List 2" can maximally receive the "rest" of list memory). Mem2 = 0 is allowed. Mem2 > $2^{23} - \text{Mem1}$ is corrected to Mem2 = $2^{23} - \text{Mem1}$. – Mem2 = "-1" is interpreted as Mem2 = $(2^{32}-1)$ and corrected to Mem2 = $2^{23} - \text{Mem1}$. Example: With <code>config_list(Mem1, -1)</code>, "List 2" is automatically assigned with the "rest" of list memory (Mem2 = $2^{23} - \text{Mem1}$, no memory for "List 3"). – Storage positions for "List 3": $2^{23} - \text{Mem1} - \text{Mem2}$. The RTC6 list memory contains 2^{23} storage positions in total. By default, it is preconfigured so that "List 1" and "List 2" can each accept 4,194,304 (= 2^{22}) list commands (Mem1 = Mem2 = 4,194,304). The protected list memory area "List 3" owns no storage positions, because $2^{23} - \text{Mem1} - \text{Mem2} = 0$. config_list is not executed (get_last_error return code RTC6_BUSY), if: <ul style="list-style-type: none"> – the BUSY list execution status is set – a list has been paused by set_wait (PAUSED list execution status set)



Ctrl Command	config_list
Comments (cont'd)	<ul style="list-style-type: none"> Configuration by <code>config_list</code> 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 has been 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. Also observe the notes in Chapter 6.3.2 "Configuring the RTC6 List Memory", page 102. If you do not know the current configuration data for list memory (<code>Mem1</code> and <code>Mem2</code>), you can find out after <code>load_list(ListNo, 0)</code> or <code>set_start_list_pos(ListNo, 0)</code> by using <code>get_list_space</code> (with "ownership" changes, <code>get_config_list</code> must be called in advance).
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality. In addition: increased value range and changed initialization values.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	get_config_list

Ctrl Command	control_command	
Function	Sends a control command. Whether and how the addressed scan system reacts depends on its properties (among other things, the scan system firmware). See also Comments, page 346 .	
Call	<code>control_command(Head, Axis, Data)</code>	
Parameters	Head	Scan head connector number of the RTC control board. As an unsigned 32-bit value. Allowed values: = 1: The scan system at 1st scan head connector. = 2: The scan system at 2nd scan head connector.
	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. See Chapter 20 "Appendix E: iDRIVE Scan Systems – Control Commands and Signals Transmitted to RTC Control Boards", page 916 . As an unsigned 32-bit value. The upper part (Bit #16...Bit #31) is <i>not</i> evaluated. The lower part (Bit #0...Bit #15) is evaluated as follows: <ul style="list-style-type: none"> • $\text{Code}_{\text{HIGH}}$ The more significant data byte (Bit #8...Bit 15). Represents a command code. Presented as hexadecimal number "(hex)" in Chapter 20 "Appendix E: iDRIVE Scan Systems – Control Commands and Signals Transmitted to RTC Control Boards", page 916. • Code_{LOW} The less significant data byte (Bit #0...Bit #7). Represents an optional parameter. Presented as hexadecimal number "(hex)" in Chapter 20 "Appendix E: iDRIVE Scan Systems – Control Commands and Signals Transmitted to RTC Control Boards", page 916. Example: By $\text{Data} = 0501_{\text{H}}$, that is, $\text{Code}_{\text{HIGH}} = 05$ (SetMode) and $\text{Code}_{\text{LOW}} = 01$ the actual position is set as the data type to be transmitted.



Ctrl Command	control_command
Comments	<ul style="list-style-type: none"> • control_command can only be used in conjunction with <i>iDRIVE</i> scan systems (see Glossary entry on page 26). • control_command is not evaluated by conventional scan systems (without <i>iDRIVE</i> technology). • control_command is not executed with unallowed values of <code>Head</code> and/or <code>Axis</code> (<code>get_last_error</code> return code <code>RTC6_PARAM_ERROR</code>). • Command code <code>Data</code> is passed to the scan system instead of the usual position data. Therefore, the corresponding galvanometer scanner Microstep is omitted, if control_command is called during execution of a list. • Under some circumstances, control_command might be unavailable at the first scan head connector if speed-dependent laser control has been activated by <code>set_auto_laser_control(Mode = 2)</code>.
RTC4→RTC6	<p>Unchanged functionality.</p> <p>Note that all data selected to be transmitted by control_command are always in the 20-bit range, even in RTC4 Compatibility Mode (see also comments on get_value).</p>
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 of the indexed character, text string or subroutine whose entries should be copied from <code>Src</code>. As an unsigned 32-bit value. Allowed value range: [0...1023] for indexed characters or subroutines, [1024+0...1024+41] for indexed text strings.</p> <p>Src Index of the indexed character, text string or subroutine whose entries should be copied to <code>Dst</code>. As an unsigned 32-bit value. Allowed value range: [0...1023] for indexed characters or subroutines, [1024+0...1024+41] for indexed text strings.</p> <p>Mode Determines which management tables are to be changed. As an unsigned 32-bit value.</p> <p>Bit #0 = 0: <code>Dst</code> is the index of an indexed character or the index of an indexed text string.</p> <p>Bit #0 = 1: <code>Dst</code> is the index of an indexed subroutine.</p> <p>Bit #1 = 0: <code>Src</code> is the index of an indexed character or the index of an indexed text string.</p> <p>Bit #1 = 1: <code>Src</code> is the index of an indexed subroutine.</p> <p>Bit #2...Bit #31: 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 <code>copy_dst_src</code> is ignored (<code>get_last_error</code> return code <code>RTC6_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). <code>copy_dst_src</code> only alters the corresponding entry in the internal management table and does not modify the content of the list memory. 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 list memory area "List 3" can be created (subsequent to <code>copy_dst_src</code>) with <code>save_disk/load_disk</code>. 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 memory overflow of the protected list memory area "List 3". See also Section "Managing Indexed Characters and Text Strings", page 119.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	save_disk , load_disk

Ctrl Command	create_dat_file
Function	Generates a new <code>RTC6DAT.dat</code> file of the current version.
Call	<code>Version = create_dat_file (Flag)</code>
Parameters	<p>Flag As a signed 32-bit value.</p> <p> 1: Generates a new <code>RTC6DAT.dat</code> file and returns the current version.</p> <p> -1: Only returns the current version.</p>
Result	<p>Version As an unsigned 32-bit value.</p> <p> > 600: Current version.</p> <p> 0: <code>create_dat_file</code> cannot be executed <code>(get_last_error</code> return code <code>RTC6_BUSY</code> or no program is loaded on the RTC6).</p> <p> 1: Not enough Windows memory <code>(get_last_error</code> return code <code>RTC6_OUT_OF_MEMORY</code>).</p> <p> 2: The output file cannot be opened <code>(get_last_error</code> return code <code>RTC6_PARAM_ERROR</code>). Make sure that you have write permissions granted for the output path containing the <code>RTC6DAT.dat</code>.</p>
Comments	<ul style="list-style-type: none"> Earlier <code>RTC6DAT.dat</code> versions cannot be used anymore. Otherwise, <code>load_program_file</code> returns error code 7 <code>(get_last_error</code> return code <code>RTC6_VERSION_MISMATCH</code>). <code>RTC6DAT.dat</code> version information is available only after <code>load_program_file</code> has also been executed with the current <code>RTC6 DLL</code> instance. <code>RTC6DAT.dat</code> version information is not taken over when acquiring an RTC6 board (no matter if it is already initialized or not). A newly generated <code>RTC6DAT.dat</code> (of a specific version) can be used at any time for all RTC6 boards that require the same version. Each newly generated <code>RTC6DAT.dat</code> contains (besides static initializations) also: <ul style="list-style-type: none"> user definable tables which have been loaded earlier by <code>load_varpolydelay</code>, <code>load_auto_laser_control</code>, <code>load_jump_table</code>, <code>load_jump_table_offset</code>, <code>load_position_control</code> a “freely definable wobble shape” which have been defined earlier by <code>set_wobble_vector</code> The user definable tables are automatically available after the next <code>load_program_file</code> whereas the “freely definable wobble shape” requires a <code>set_wobble_mode(Mode > 1)</code> call in addition. Note that <code>create_dat_file</code> overwrites an already existing <code>RTC6DAT.dat</code> without warning. Keep a copy of your original <code>RTC6DAT.dat</code> in a safe place. <code>create_dat_file</code> cannot be executed, while a list is being processed <code>(get_last_error</code> return code <code>RTC6_BUSY</code>).

Ctrl Command	create_dat_file
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 610, OUT 610, RBF 615. Last change DAT 603.
References	load_program_file

Ctrl Command	disable_laser
Function	Disables the Signals for "Laser Active" Operation .
Call	<code>disable_laser()</code>
Comments	<ul style="list-style-type: none"> • disable_laser disables the Signals for "Laser Active" Operation at the output ports LASER1, LASER2 and LASERON (the signals are set to their respective "Off" level). Pin (02) of the 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 pin (01) and pin (09) of the LASER Connector and pins (19) and (22) of the EXTENSION 2 socket connector are at a fixed level after disable_laser. • The 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 RTC6 with load_program_file, the laser control is deactivated and absolutely requires set_laser_control for activation. • If disable_laser results in an RTC6_TIMEOUT error (for example, if no program file has been loaded), the LASER1, LASER2 and LASERON output ports can only be reactivated with set_laser_control after a load_program_file command. • By get_startstop_info (Bit #9), the current status of the laser control signals (globally enabled, yes or no) can be queried. • By get_startstop_info (Bit #14), the status "laser enabled" (yes or no) can be queried.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600. Last change: DLL 617, OUT 617: get_startstop_info (Bit #14).
References	enable_laser , set_laser_control , get_startstop_info



Ctrl Command	<code>enable_laser</code>
Function	Enables the Signals for "Laser Active" Operation.
Call	<code>enable_laser()</code>
Comments	<ul style="list-style-type: none"> After a hardware reset (and after <code>load_program_file</code>), <code>set_laser_control</code> must be called to activate the LASER1, LASER2 and LASERON output ports and define the signal level, see Chapter 7.4 "Laser Control", page 183. Otherwise, <code>enable_laser</code> has no effect. After initialization of the RTC6 with <code>load_program_file</code>, the laser control signals are <i>deactivated</i>. For first-time activation, <code>set_laser_control</code> must be called (see above). After a subsequent disabling by <code>disable_laser</code> (or <code>set_laser_control</code>), the <code>enable_laser</code> (or <code>set_laser_control</code>) command can be used for reenabling. Even if the laser control signals have been enabled with <code>enable_laser</code> or <code>set_laser_control</code>, they are not outputted without further commands, see Chapter 7.4 "Laser Control", page 183. By <code>get_startstop_info</code> (Bit #9) the current status of the laser control signals (globally enabled, yes or no) can be queried. By <code>get_startstop_info</code> (Bit #14), the status "laser enabled" can be queried.
RTC4→RTC6	Basically unchanged functionality. In some circumstances, <code>set_laser_control</code> must be called before <code>enable_laser</code> (see above).
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	<code>disable_laser</code> , <code>set_laser_control</code> , <code>get_startstop_info</code> , <code>set_laser_mode</code>

Ctrl Command	eth_assign_card
Function	Enters the RTC6 Ethernet Board with index <code>SearchNo</code> from the search result list at the RTC6 board management index <code>CardNo</code> .
Call	<code>Result = eth_assign_card(SearchNo, CardNo)</code>
Parameters	<p><code>SearchNo</code> Index of the search result list. As an unsigned 32-bit value. Allowed value range: [1...<code>(Result value > 0 of eth_found_cards)</code>].</p> <p><code>CardNo</code> Index of the RTC6 board management, at which the RTC6 Ethernet Board is to be entered. As an unsigned 32-bit value. Allowed value range: [<code>(rtc6_count_cards + 1)</code>...255] or 0.</p>
Result	<p>Error code. As a signed 32-bit value.</p> <ul style="list-style-type: none"> -2 Error: the entry cannot be made. At this index, already an RTC6 Ethernet Board is entered. -1 Error: the entry cannot be made. At this index, already an RTC6 PCIe Board is entered. 0 Error: the entry cannot be made. <code>SearchNo</code> is invalid (0, > <code>eth_found_cards</code>). Or: <code>CardNo</code> is invalid ($1 \leq \text{CardNo} \leq \text{rtc6_count_cards}$, > 255). $n (= \text{CardNo})$ Success: the card number entry has been made.
Comments	<ul style="list-style-type: none"> • <code>eth_assign_card</code> is not available as a multi-board command. • To create a search result list, <code>eth_search_cards</code> and <code>eth_search_cards_range</code> are used. See also Chapter 16.5.2 "About Searching RTC6 Ethernet Boards", page 886. • No RTC6 Ethernet Board or RTC6 PCIe Board must already be entered at the specified index <code>CardNo</code>. An RTC6 Ethernet Board must be explicitly removed by <code>eth_remove_card</code> in advance. RTC6 PCIe Boards cannot be removed. • When successful, <code>eth_count_cards</code> is automatically increased by 1. • With <code>CardNo = 0</code> the RTC6 Ethernet Board is entered at <code>rtc6_count_cards + eth_count_cards + 1</code>. Thus, a continuous card numbering without gaps can automatically be created (see also <code>eth_remove_card</code>). <code>CardNo = 0</code> and freely chosen assignment of RTC6 Ethernet Boards to indexes should not be concurrently used. • With all errors, the <code>get_last_error</code> return code <code>RTC6_PARAM_ERROR</code> is generated. • See also Chapter 16.5.3 "About the RTC6 Board Management", page 887.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 606, OUT 606, RBF 611.
References	eth_assign_card_ip , eth_count_cards , eth_found_cards , eth_max_card , eth_search_cards , eth_search_cards_range , eth_remove_card , rtc6_count_cards , eth_set_com_timeouts_auto

Ctrl Command	eth_assign_card_ip
Function	Enters an RTC6 Ethernet Board with the IP address <code>Ip</code> at the index <code>CardNo</code> in the RTC6 board management.
Call	<code>Result = eth_assign_card_ip(Ip, CardNo)</code>
Parameters	<code>Ip</code> IP address for the RTC6 Ethernet Board. As an unsigned 32-bit value. <code>CardNo</code> Index of the RTC6 board management, at which the RTC6 Ethernet Board is to be entered. As an unsigned 32-bit value. Allowed value range: [<code>(rtc6_count_cards</code> + 1)...255] or 0.
Result	Error code. As a signed 32-bit value. -2 Error: the entry cannot be made. At this index, already an RTC6 Ethernet Board is entered. -1 Error: the entry cannot be made. At this index, already an RTC6 PCIe Board is entered. 0 Error: the entry cannot be made. CardNo is invalid ($1 \leq \text{CardNo} \leq \text{rtc6_count_cards}$, > 255). <code>n</code> (= <code>CardNo</code>) Success: the entry has been made.
Comments	<ul style="list-style-type: none"> • <code>eth_assign_card_ip</code> is not available as a multi-board command. • No RTC6 Ethernet Board or RTC6 PCIe Board must already be entered at the specified index <code>CardNo</code>. An RTC6 Ethernet Board must be explicitly removed by <code>eth_remove_card</code> in advance. RTC6 PCIe Boards cannot be removed. • When successful, <code>eth_count_cards</code> is automatically increased by 1. • With <code>CardNo</code> = 0 the RTC6 Ethernet Board is entered at <code>rtc6_count_cards</code> + <code>eth_count_cards</code> + 1. Thus, a continuous card numbering without gaps can automatically be created (see also <code>eth_remove_card</code>). <code>CardNo</code> = 0 and freely chosen assignment of RTC6 Ethernet Boards to indexes should not be concurrently used. • With all errors, the <code>get_last_error</code> return code <code>RTC6_PARAM_ERROR</code> is generated. • The RTC6 Ethernet Board is entered in the RTC6 board management, even if there is an Ethernet error during its initialization. Before trying to initialize once again, this RTC6 Ethernet Board must be removed explicitly from the RTC6 board management. • See also Chapter 16.5.3 "About the RTC6 Board Management", page 887.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 606, OUT 606, RBF 611.
References	eth_assign_card , eth_count_cards , eth_remove_card , eth_set_com_timeouts_auto

Ctrl Command	eth_boot_dcmsg
Function	Standalone Functionality: Defines the next following control command as a boot command for Standalone Operation Mode .
Prerequisite	RTC6 Software Package \geq V1.7.0 and BIOS-ETH \geq 26.
Call	<code>eth_boot_dcmsg()</code>
Parameters	None.
Result	None.
Comments	<ul style="list-style-type: none"> The definition refers to the subsequent control command, that is, list commands in between do not matter. The boot definition is omitted without replacement, if the defined control command is not permitted for booting, see Chapter 16.7.5 "Control Commands Allowed for Automatic Booting", page 894. eth_boot_dcmsg is only allowed with RTC6 Ethernet Boards. Otherwise, a <code>get_last_error</code> return code <code>RTC6_TYPE_REJECTED</code> is generated. See Chapter 16.7 "Standalone Functionality", page 890.
RTC4 \rightarrow RTC6	New command.
RTC5 \rightarrow RTC6	New command.
Version info	Available as of DLL 618, OUT 618, RBF 623.
References	eth_boot_timeout

Ctrl Command	eth_boot_timeout
Function	Standalone Functionality: Sets the waiting time for the automatic transition from Boot Phase 1 to Boot Phase 2 .
Prerequisite	RTC6 Software Package \geq V1.7.0 and BIOS-ETH \geq 26.
Call	<code>eth_set_search_cards_timeout(TimeOut)</code>
Parameters	TimeOut Waiting time. In s. As an unsigned 32-bit value.
Result	None.
Comments	<ul style="list-style-type: none"> eth_boot_timeout is only allowed with RTC6 Ethernet Boards. Otherwise, a <code>get_last_error</code> return code <code>RTC6_TYPE_REJECTED</code> is generated. TimeOut = 0 deactivates the waiting time. The boot process waits endlessly for an /START. Boot Phase 2 starts immediately when an /START is triggered within the waiting time. See Chapter 16.7 "Standalone Functionality", page 890.
RTC4 \rightarrow RTC6	New command.
RTC5 \rightarrow RTC6	New command.
Version info	Available as of DLL 618, OUT 618, RBF 623.
References	eth_boot_dcmsg



Ctrl Command	eth_check_connection
Function	Checks whether the RTC6 Ethernet Board responds.
Call	<code>CheckOK = eth_check_connection()</code>
Parameters	None.
Result	<p>Response behavior. As an unsigned 32-bit value.</p> <p>0: Not OK. The RTC6 Ethernet Board does not have an Ethernet connection, it does not respond or it is not an RTC6 Ethernet Board.</p> <p>1: OK. The RTC6 Ethernet Board has an Ethernet connection and it responds.</p>
Comments	<ul style="list-style-type: none"> When <code>CheckOK = 0</code>, the possible error cause can be queried by get_last_error, eth_get_last_error and eth_get_error. If the board is not an RTC6 Ethernet Board, then: <ul style="list-style-type: none"> <code>eth_check_connection</code> is not executed <code>get_last_error</code> return code <code>RTC6_TYPE_REJECTED</code> is set See also Chapter 16.5.4 "Checking the Connection to the RTC6 Ethernet Board", page 888.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 608, OUT 608, RBF 611.
References	eth_get_error , eth_get_last_error , get_last_error



Ctrl Command	eth_configure_link_loss	
Function	Sets the behavior of the RTC6 Ethernet Board in case of an Ethernet Link Loss .	
Prerequisite	RTC6 Software Package \geq V1.7.6 and BIOS-ETH \geq 28.	
Call	eth_configure_link_loss(Mode)	
Parameters	Mode	<p>Action to take. As an unsigned 32-bit value.</p> <p>0: The Signals for "Laser Active" Operation are suppressed immediately. List execution continues.</p> <p>1: The Signals for "Laser Active" Operation are switched-off immediately. List execution is stopped immediately (as with stop_execution or /STOP).</p> <p>2: A simulate_ext_stop is passed on to all slave boards.</p> <p>3: Ethernet Link Loss detection switched off (default after load_program_file).</p>
Result	None.	
Comments	<ul style="list-style-type: none"> For Mode $>$ 3, the get_last_error return code RTC6_PARAM_ERROR is set and eth_configure_link_loss is not executed. The RTC6 Ethernet Board detects an Ethernet Link Loss after 100 ms at the latest. Whether an Ethernet Link Loss has been detected can be queried by get_startstop_info. 	
RTC4→RTC6	New command.	
RTC5→RTC6	New command.	
Version info	Available as of DLL 622, OUT 622.	
References	get_startstop_info	



Ctrl Command	eth_convert_ip_to_string				
Function	Converts an IP address in big-endian byte order to the usual dotted decimal notation (for example, "192.168.250.1").				
Call	<code>eth_convert_ip_to_string(Ip, IpString)</code>				
Parameters	<table> <tr> <td><code>Ip</code></td><td>To-be converted IP address in big-endian byte order. As an unsigned 32-bit value.</td></tr> <tr> <td><code>IpString</code></td><td>Converted IP address. As a pointer (in C and C++ data type <code>ULONG_PTR</code>, an unsigned 32-bit value or unsigned 64-bit value) to an array of 4 unsigned 32-bit values, where the converted IP address as a string in usual dotted decimal notation can be found (synonymously to a character array of length 16, low byte first, with concluding <code>\0</code>).</td></tr> </table>	<code>Ip</code>	To-be converted IP address in big-endian byte order. As an unsigned 32-bit value.	<code>IpString</code>	Converted IP address. As a pointer (in C and C++ data type <code>ULONG_PTR</code> , an unsigned 32-bit value or unsigned 64-bit value) to an array of 4 unsigned 32-bit values, where the converted IP address as a string in usual dotted decimal notation can be found (synonymously to a character array of length 16, low byte first, with concluding <code>\0</code>).
<code>Ip</code>	To-be converted IP address in big-endian byte order. As an unsigned 32-bit value.				
<code>IpString</code>	Converted IP address. As a pointer (in C and C++ data type <code>ULONG_PTR</code> , an unsigned 32-bit value or unsigned 64-bit value) to an array of 4 unsigned 32-bit values, where the converted IP address as a string in usual dotted decimal notation can be found (synonymously to a character array of length 16, low byte first, with concluding <code>\0</code>).				
Result	None.				
Comments	<ul style="list-style-type: none"> • eth_convert_ip_to_string is not available as a multi-board command. • For storage of the converted IP address, the user program must provide a memory area of 4×4 bytes at the address specified by <code>IpString</code>. • Example: <code>Ip</code> = 33204416, <code>IpString[0]</code> = "192.", <code>IpString[1]</code> = "168.", <code>IpString[2]</code> = "250.", <code>IpString[3]</code> = "1\0xx". • The use of eth_convert_ip_to_string is useful, for example, for eth_get_static_ip. • See also Chapter 16.5.1 "Notes on Working with IP Addresses", page 886. 				
RTC4→RTC6	New command.				
RTC5→RTC6	New command.				
Version info	Available as of DLL 606, OUT 606, RBF 611.				
References	eth_convert_string_to_ip , eth_get_static_ip				



Ctrl Command	eth_convert_string_to_ip
Function	Converts an IP address in usual dotted decimal notation (for example, "192.168.250.1") to the big-endian byte order.
Call	<code>ResultIp = eth_convert_string_to_ip(&IpString)</code>
Parameters	<p>IpString To-be converted IP address in usual dotted decimal notation. As a string (char array). As a pointer to a \0-terminated ANSI string. 16 bytes max.</p>
Result	<p>IP address. As an unsigned 32-bit value.</p> <p>0: IpString is faulty.</p> <p>> 0: Converted IP address in big-endian byte order.</p>
Comments	<ul style="list-style-type: none"> • eth_convert_string_to_ip is not available as a multi-board command. • Example: With parameter <code>IpString = "192.168.250.1"</code> the result value is <code>IpString = 33204416</code>. See also comment at eth_search_cards. • The use of eth_convert_string_to_ip is useful, for example, for eth_search_cards, eth_search_cards_range and eth_set_static_ip as well as with eth_get_card_info and eth_get_card_info_search. • See also Chapter 16.5.1 "Notes on Working with IP Addresses", page 886.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 606, OUT 606, RBF 611.
References	eth_convert_ip_to_string , eth_get_card_info , eth_get_card_info_search , eth_search_cards , eth_search_cards_range , eth_set_static_ip

Ctrl Command	eth_count_cards
Function	Returns the number of RTC6 Ethernet Boards entered in the <i>RTC6 board management</i> .
Call	Result = eth_count_cards()
Parameters	None.
Result	Number of RTC6 Ethernet Boards recorded in the RTC6 board management. As an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> • eth_count_cards is not available as a multi-board command. • Unlike rtc6_count_cards, you cannot deduce in each case the highest index of all RTC6 boards from the result value (see eth_assign_card and eth_remove_card with CardNo = 0, see also eth_max_card). • It is only ensured that each RTC6 Ethernet Board number is greater than the highest RTC6 PCIe Board number, see rtc6_count_cards. • Certainly eth_count_cards can be greater than eth_found_cards. This is the case, if RTC6 Ethernet Boards have been entered explicitly by its IP addresses into the RTC6 board management, see eth_assign_card_ip. • After successful execution of eth_assign_card, eth_count_cards is automatically increased by 1. • After successful execution of eth_remove_card, eth_count_cards is automatically decreased by 1. • See also Chapter 16.5.3 "About the RTC6 Board Management", page 887.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 606, OUT 606, RBF 611.
References	eth_assign_card , eth_assign_card_ip , eth_found_cards , eth_max_card , eth_remove_card , rtc6_count_cards

Ctrl Command	eth_found_cards
Function	Returns the number of RTC6 Ethernet Boards in the search result list.
Call	Result = eth_found_cards()
Parameters	None.
Result	<p>Number of RTC6 Ethernet Boards in the search result list. As an unsigned 32-bit value.</p> <p>0: There are no RTC6 Ethernet Boards in the search result list. Possibly the search has not been carried-out yet.</p> <p>n: Number of RTC6 Ethernet Boards contained in the search result list.</p>
Comments	<ul style="list-style-type: none"> • eth_found_cards is not available as a multi-board command. • eth_found_cards does not necessarily match the <ul style="list-style-type: none"> – RTC6 Ethernet Board number in the RTC6 board management – total number of RTC6 Ethernet Boards and RTC6 PCIe Boards. • To create the search result list, use either eth_search_cards or eth_search_cards_range. Their result value is the same as the one of eth_found_cards. See also Chapter 16.5.2 "About Searching RTC6 Ethernet Boards", page 886. • Information about a particular RTC6 Ethernet Board in the search result list can be queried by eth_get_card_info_search. • See also Chapter 16.5.3 "About the RTC6 Board Management", page 887.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 606, OUT 606, RBF 611.
References	eth_get_card_info_search , eth_search_cards , eth_search_cards_range



Ctrl Command	eth_get_card_info
Function	Returns information about a particular RTC6 Ethernet Board enlisted in the RTC6 board management and has been previously acquired once or is currently acquired.
Call	<code>eth_get_card_info(CardNo, Ptr)</code>
Parameters	<p>CardNo Index of the RTC6 Ethernet Board in the RTC6 board management. As an unsigned 32-bit value.</p> <p>Ptr Card information. As a pointer (in C and C++ data type <code>ULONG_PTR</code>, an unsigned 32-bit value or unsigned 64-bit value) to an array of 16 unsigned 32-bit values.</p>
Result	None.
Comments	<ul style="list-style-type: none"> • <code>eth_get_card_info</code> is not available as a multi-board command. • For storage of the card information, the user program must provide (at the address specified by <code>Ptr</code>) a memory area of 16×4 bytes. • Card information: <ul style="list-style-type: none"> [0] = Firmware version of the Ethernet communication software [1] = Serial number [2] = IP address (Big Endian format) [3] = MAC address (lower 4 bytes) [4] = MAC address (upper 2 bytes) [5] = Is acquired (1 or 0) [6] = IP address of the connected PC (Big Endian format) [7] = Force DHCP (1 or 0) [8] = Static IP address (Big Endian format) [9] = Static net mask (Big Endian format) [10] = Static gateway (Big Endian format) [11] = UDP port for board searches [12] = UDP port for exclusive connections [13] = TCP port for an exclusive connection [14] = Reserved [15] = Reserved • See also Chapter 16.5.3 "About the RTC6 Board Management", page 887.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 606, OUT 606, RBF 611.
References	eth_count_cards , eth_get_card_info_search , eth_max_card , get_card_type



Ctrl Command	eth_get_card_info_search
Function	Returns information about a particular RTC6 Ethernet Board in the search result list.
Call	<code>eth_get_card_info_search(SearchNo, Ptr)</code>
Parameters	<p>SearchNo Index of the search result list. As an unsigned 32-bit value.</p> <p>Ptr Card information. As a pointer (in C and C++ data type ULONG_PTR, an unsigned 32-bit value or unsigned 64-bit value) to an array of 16 unsigned 32-bit values.</p>
Result	None.
Comments	<ul style="list-style-type: none"> • eth_get_card_info_search is not available as a multi-board command. • To create the search result list, use either eth_search_cards or eth_search_cards_range. Their result value is the same as the one of eth_found_cards. See also Chapter 16.5.2 "About Searching RTC6 Ethernet Boards", page 886. • For <code>SearchNo = 0</code> and <code>SearchNo > eth_found_cards</code> a default information is returned. Furthermore, the get_last_error return code RTC6_PARAM_ERROR is generated. • If an search has not yet been carried-out then automatically the following applies: <code>SearchNo > eth_found_cards</code>. • For storage of the card information, the user program must provide (at the address specified by <code>Ptr</code>) a memory area of 16×4 bytes. • Card information: Same as eth_get_card_info.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 606, OUT 606, RBF 611.
References	eth_found_cards , eth_get_card_info , eth_search_cards , eth_search_cards_range

Ctrl Command	eth_get_com_timeouts
Function	Returns timing information for an RTC6 Ethernet Board that is entered in the RTC6 board management.
Call	<code>eth_get_com_timeouts(&AcquireTimeout, &AcquireMaxRetries, &SendRecvTimeout, &SendRecvMaxRetries, &KeepAlive, &KeepInterval)</code>
Result	None.
Returned parameter values	<p>AcquireTimeout As a pointer to an unsigned 32-bit value. AcquireMaxRetries See AcquireTimeout. SendRecvTimeout See AcquireTimeout. SendRecvMaxRetries See AcquireTimeout. KeepAlive See AcquireTimeout. KeepInterval See AcquireTimeout.</p> <p>All values are each 0, if the board is not an RTC6 Ethernet Board. All time specifications in μs.</p>
Comments	<ul style="list-style-type: none"> See also Chapter 16.5.3 "About the RTC6 Board Management", page 887. By eth_set_com_timeouts, the settings can be changed (usually only in the context of a problem clarification and only on request by SCANLAB). The parameters <code>AcquireTimeout</code>, <code>AcquireMaxRetries</code>, <code>SendRecvTimeout</code> and <code>SendRecvMaxRetries</code> refer only to the RTC6 DLL. Therefore, they can be read out even without granted access right to an RTC6 Ethernet Board. The returned parameters <code>KeepAlive</code> and <code>KeepInterval</code> have the value 0, if for the addressed RTC6 Ethernet Board <ul style="list-style-type: none"> – there is no access right granted (\leq DLL 618: get_last_error return code <code>RTC6_ACCESS_DENIED</code> and <code>RTC6_ETH_ERROR</code>; \geq DLL 619: no error message any more) – there is a granted access right, but no connection is established (get_last_error return code <code>RTC6_ETH_ERROR</code>)
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 606, OUT 606, RBF 611. Last change DLL 619: no error message any more, if no access right granted.
References	eth_set_com_timeouts , eth_set_com_timeouts_auto

Ctrl Command	eth_get_com_timeouts_auto	
Function	Returns timing information (for an automatic mechanism) for an RTC6 Ethernet Board that is entered in the RTC6 board management.	
Call	eth_get_com_timeouts_auto(&InitialTimeout, &MaxTimeout, &Multiplier, &Mode)	
Result	None.	
Returned parameter values	InitialTimeout	The timeout to start with. In ms. As a pointer to a 64-bit IEEE floating point value.
	MaxTimeout	Maximum timeout (= sum of the timeouts of the individual attempts). After MaxTimeout has elapsed, the network connection is considered to be faulty (udp_recv_timeout error, see eth_get_last_error). In ms. As a pointer to a 64-bit IEEE floating point value.
	Multiplier	Multiplication factor by which the timeout is increased for each attempt. As a pointer to a 64-bit IEEE floating point value.
	Mode	Reserved. As a pointer to an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> See also Chapter 16.5.3 "About the RTC6 Board Management", page 887. By eth_set_com_timeouts_auto, the settings can be changed. eth_get_com_timeouts_auto returns the values of the automatic mechanism (for setting appropriate timeout values). All time specifications in ms, not μs. Default values after eth_assign_card or eth_assign_card_ip: InitialTimeout 0.75, MaxTimeout 20.0, Multiplier 1.3 The parameter Mode is reserved for future extensions. Returns 1. eth_get_com_timeouts_auto affects solely the RTC6 DLL. There is no need for a connection to the RTC6 Ethernet Board. 	
RTC4→RTC6	New command.	
RTC5→RTC6	New command.	
Version info	Available as of DLL 620, OUT 620.	
References	eth_set_com_timeouts_auto , eth_get_com_timeouts	

Ctrl Command	eth_get_error
Function	Returns an accumulated error code. It contains all errors which occurred with the most recently executed RTC6 Ethernet Board commands.
Call	<code>EthAccError = eth_get_error()</code>
Parameters	None.
Result	Error code. Identical to eth_get_last_error . As an unsigned 32-bit value. If multiple errors occurred simultaneously, then multiple bits are set.
Comments	<ul style="list-style-type: none"> For error handling, see Chapter 6.8 "Error Handling", page 129. eth_get_error and n_eth_get_error are available even without explicit access rights to a specific RTC6 Ethernet Board. The board-specific error variables <code>LastError</code> and <code>AccError</code> (see Chapter 6.8 "Error Handling", page 129) are neither generated nor altered by eth_get_error. If with an RTC6 Ethernet Board command call (internally) several errors occur successively, eth_get_last_error probably returns only the very last error (for example, only "not_acquired"). The actual error causes can be identified by eth_get_error because it returns the accumulated errors. The accumulated error is reset automatically with each acquiring of the RTC6 Ethernet Board.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 608, OUT 608, RBF 611.
References	acquire_rtc , eth_get_last_error

Ctrl Command	eth_get_ip
Function	Returns the pertaining IP address for a specified RTC6 board management index.
Call	IP = eth_get_ip(CardNo)
Parameters	CardNo Index of the RTC6 Ethernet Board in the RTC6 board management. As an unsigned 32-bit value.
Result	IP address. In Big Endian byte order. The value is 0, if the board is not an RTC6 Ethernet Board.
Comments	<ul style="list-style-type: none"> • eth_get_ip is not available as a multi-board command. • To convert from Big Endian byte order to the usual dotted decimal notation, eth_convert_ip_to_string can be used. See also Chapter 16.5.1 "Notes on Working with IP Addresses", page 886. • eth_get_ip allows a faster IP address query from the RTC6 board management than by eth_get_card_info.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 606, OUT 606, RBF 611.
References	eth_convert_ip_to_string , eth_get_card_info

Ctrl Command	eth_get_ip_search
Function	Returns the pertaining IP address for a specified RTC6 Ethernet Board in the search result list.
Call	IP = eth_get_ip_search(SearchNo)
Parameters	SearchNo Index in the search result list. As an unsigned 32-bit value.
Result	IP address. In Big Endian byte order.
Comments	<ul style="list-style-type: none"> • eth_get_ip_search is not available as a multi-board command. • To create the search result list, use either eth_search_cards or eth_search_cards_range. Their result value is the same as the one of eth_found_cards. See also Chapter 16.5.2 "About Searching RTC6 Ethernet Boards", page 886. • The result value is 0 for SearchNo = 0 and SearchNo > eth_found_cards. At the same time the get_last_error return code RTC6_PARAM_ERROR is generated. • If an search has not yet been carried-out then automatically the following applies: SearchNo > eth_found_cards.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 608, OUT 608, RBF 611.
References	eth_get_ip

Ctrl Command	eth_get_last_error																																																																																									
Function	Returns an error code which contains only the errors that have occurred during execution of the most recent RTC6 Ethernet Board command.																																																																																									
Call	EthLastError = eth_get_last_error()																																																																																									
Parameters	None.																																																																																									
Result	<p>Error code. Identical to eth_get_error. As an unsigned 32-bit value. If several errors occurred at the same time, several bits are set.</p> <table> <thead> <tr> <th>no_error = 0. No error.</th> <th>Error</th> <th>Numeric value ($2^{[Bit]}$)</th> </tr> </thead> <tbody> <tr><td>wsasstartup_failed</td><td>Bit 0</td><td>1</td></tr> <tr><td>wrong_winsock_version</td><td>Bit 1</td><td>2</td></tr> <tr><td>Reserved.</td><td>Bit 2</td><td>4</td></tr> <tr><td>udp_create_socket_error</td><td>Bit 3</td><td>8</td></tr> <tr><td>udp_bind_socket_error</td><td>Bit 4</td><td>16</td></tr> <tr><td>udp_connect_socket_error</td><td>Bit 5</td><td>32</td></tr> <tr><td>udp_excl_add_use_error</td><td>Bit 6</td><td>64</td></tr> <tr><td>udp_bc_ena_error</td><td>Bit 7</td><td>128</td></tr> <tr><td>Reserved.</td><td>Bit 8</td><td>256</td></tr> <tr><td>tcp_create_socket_error</td><td>Bit 9</td><td>512</td></tr> <tr><td>tcp_excl_add_use_error</td><td>Bit 10</td><td>1024</td></tr> <tr><td>tcp_bind_socket_error</td><td>Bit 11</td><td>2048</td></tr> <tr><td>Reserved.</td><td>Bit 12</td><td>4096</td></tr> <tr><td>tcp_connect_socket_error</td><td>Bit 13</td><td>8192</td></tr> <tr><td>tcp_connect_sel_error</td><td>Bit 14</td><td>16384</td></tr> <tr><td>tcp_connect_fds_error</td><td>Bit 15</td><td>32768</td></tr> <tr><td>tcp_nodelay_error</td><td>Bit 16</td><td>65536</td></tr> <tr><td>create_thread_failed</td><td>Bit 17</td><td>131072</td></tr> <tr><td>udp_recv_error</td><td>Bit 18</td><td>262144</td></tr> <tr><td>udp_send_error</td><td>Bit 19</td><td>524288</td></tr> <tr><td>udp_recv_timeout</td><td>Bit 20</td><td>1048576</td></tr> <tr><td>already_acquired</td><td>Bit 21</td><td>2097152</td></tr> <tr><td>not_acquired</td><td>Bit 22</td><td>4194304</td></tr> <tr><td>access_denied</td><td>Bit 23</td><td>8388608</td></tr> <tr><td>send_tgm_timeout</td><td>Bit 24</td><td>16777216</td></tr> <tr><td>card_not_found</td><td>Bit 25</td><td>33554432</td></tr> <tr><td>core1_timeout</td><td>Bit 26</td><td>67108864</td></tr> <tr><td>bootmode_set_failed</td><td>Bit 27</td><td>134217728</td></tr> </tbody> </table>			no_error = 0. No error.	Error	Numeric value ($2^{[Bit]}$)	wsasstartup_failed	Bit 0	1	wrong_winsock_version	Bit 1	2	Reserved.	Bit 2	4	udp_create_socket_error	Bit 3	8	udp_bind_socket_error	Bit 4	16	udp_connect_socket_error	Bit 5	32	udp_excl_add_use_error	Bit 6	64	udp_bc_ena_error	Bit 7	128	Reserved.	Bit 8	256	tcp_create_socket_error	Bit 9	512	tcp_excl_add_use_error	Bit 10	1024	tcp_bind_socket_error	Bit 11	2048	Reserved.	Bit 12	4096	tcp_connect_socket_error	Bit 13	8192	tcp_connect_sel_error	Bit 14	16384	tcp_connect_fds_error	Bit 15	32768	tcp_nodelay_error	Bit 16	65536	create_thread_failed	Bit 17	131072	udp_recv_error	Bit 18	262144	udp_send_error	Bit 19	524288	udp_recv_timeout	Bit 20	1048576	already_acquired	Bit 21	2097152	not_acquired	Bit 22	4194304	access_denied	Bit 23	8388608	send_tgm_timeout	Bit 24	16777216	card_not_found	Bit 25	33554432	core1_timeout	Bit 26	67108864	bootmode_set_failed	Bit 27	134217728
no_error = 0. No error.	Error	Numeric value ($2^{[Bit]}$)																																																																																								
wsasstartup_failed	Bit 0	1																																																																																								
wrong_winsock_version	Bit 1	2																																																																																								
Reserved.	Bit 2	4																																																																																								
udp_create_socket_error	Bit 3	8																																																																																								
udp_bind_socket_error	Bit 4	16																																																																																								
udp_connect_socket_error	Bit 5	32																																																																																								
udp_excl_add_use_error	Bit 6	64																																																																																								
udp_bc_ena_error	Bit 7	128																																																																																								
Reserved.	Bit 8	256																																																																																								
tcp_create_socket_error	Bit 9	512																																																																																								
tcp_excl_add_use_error	Bit 10	1024																																																																																								
tcp_bind_socket_error	Bit 11	2048																																																																																								
Reserved.	Bit 12	4096																																																																																								
tcp_connect_socket_error	Bit 13	8192																																																																																								
tcp_connect_sel_error	Bit 14	16384																																																																																								
tcp_connect_fds_error	Bit 15	32768																																																																																								
tcp_nodelay_error	Bit 16	65536																																																																																								
create_thread_failed	Bit 17	131072																																																																																								
udp_recv_error	Bit 18	262144																																																																																								
udp_send_error	Bit 19	524288																																																																																								
udp_recv_timeout	Bit 20	1048576																																																																																								
already_acquired	Bit 21	2097152																																																																																								
not_acquired	Bit 22	4194304																																																																																								
access_denied	Bit 23	8388608																																																																																								
send_tgm_timeout	Bit 24	16777216																																																																																								
card_not_found	Bit 25	33554432																																																																																								
core1_timeout	Bit 26	67108864																																																																																								
bootmode_set_failed	Bit 27	134217728																																																																																								



Ctrl Command	eth_get_last_error
Comments	<ul style="list-style-type: none"> For error handling, Chapter 6.8 "Error Handling", page 129. eth_get_last_error and n_eth_get_last_error are available even without explicit access rights to a specific RTC6 Ethernet Board. The board-specific error variables <code>LastError</code> and <code>AccError</code> (Chapter 6.8 "Error Handling", page 129) are neither generated nor altered by eth_get_last_error. Each eth_get_last_error error also leads to a get_last_error error <code>RTC6_ETH_ERROR</code>. If with an RTC6 Ethernet Board command call (internally) several errors occur successively, eth_get_last_error probably returns only the very last error (for example, only "<code>not_acquired</code>"). The actual error causes can be identified by eth_get_error because it returns the accumulated errors.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 606, OUT 606, RBF 611.
References	eth_get_error , get_error , reset_error , set_verify

Ctrl Command	eth_get_port_numbers	
Function	Returns the UDP ports for board searches (UDPsearch) and exclusive connections (UDPexcl) as well as the TCP port of an RTC6 Ethernet Board.	
Call	Result = eth_get_port_numbers(&UDPsearch, &UDPexcl, &TCP)	
Result	Error code. As an unsigned 32-bit value. 0: Success: OK. 1: Error: Not an RTC6 Ethernet Board.	
Returned parameter values	UDPsearch	UDP port for board searches. As a pointer to an unsigned 32-bit value.
	UDPexcl	UDP port for exclusive connections. As a pointer to an unsigned 32-bit value.
	TCP	TCP port for exclusive connections. As a pointer to an unsigned 32-bit value. All values are each 0, if the board is not an RTC6 Ethernet Board.
Comments	<ul style="list-style-type: none"> The values can also be queried by eth_get_card_info. 	
RTC4→RTC6	New command.	
RTC5→RTC6	New command.	
Version info	Available as of DLL 606, OUT 606, RBF 611.	
References	eth_get_card_info , eth_set_port_numbers	

Ctrl Command	eth_get_serial_search	
Function	Returns the pertaining serial number for a specified RTC6 Ethernet Board in the search result list.	
Call	Serialnumber = eth_get_serial_search(SearchNo)	
Parameters	SearchNo Index in the search result list. As an unsigned 32-bit value.	
Result	Serial number. As an unsigned 32-bit value.	
Comments	<ul style="list-style-type: none"> eth_get_serial_search is not available as a multi-board command. To create the search result list, use either eth_search_cards or eth_search_cards_range. Their result value is the same as the one of eth_found_cards. See also Chapter 16.5.2 "About Searching RTC6 Ethernet Boards", page 886. The result value is 0 for SearchNo = 0 and SearchNo > eth_found_cards. At the same time the get_last_error return code RTC6_PARAM_ERROR is generated. If an search has not yet been carried-out then automatically the following applies: SearchNo > eth_found_cards. 	
RTC4→RTC6	New command.	
RTC5→RTC6	New command.	
Version info	Available as of DLL 608, OUT 608, RBF 611.	
References	get_serial_number , eth_get_card_info_search	



Ctrl Command	eth_get_standalone_status						
Function	Returns Standalone Operation Mode related information about the RTC6 Ethernet Board.						
Prerequisite	RTC6 Software Package \geq V1.9.0 and BIOS-ETH \geq 30 .						
Call	<code>eth_get_standalone_status(&Status, &Error, &Mode)</code>						
Result	None.						
Parameters	None.						
Returned parameter values	<table> <tr> <td>Status</td> <td>Status of automatic booting, see also Chapter 16.7.6 "Automatic Booting – Process in Detail", page 896. As a pointer to an unsigned 32-bit value. = 0: Finished. = 1: In progress.</td> </tr> <tr> <td>Error</td> <td>Errors during automatic booting. As a pointer to an unsigned 32-bit value. = 0: No. = 1: Yes.</td> </tr> <tr> <td>Mode</td> <td>Target state after automatic booting (after a Hardware reset). As a pointer to an unsigned 32-bit value. = 0: "Normal PC Operation State". = 1: "Standalone Basic State". = 2: "Standalone Full State".</td> </tr> </table>	Status	Status of automatic booting, see also Chapter 16.7.6 "Automatic Booting – Process in Detail", page 896 . As a pointer to an unsigned 32-bit value. = 0: Finished. = 1: In progress.	Error	Errors during automatic booting. As a pointer to an unsigned 32-bit value. = 0: No. = 1: Yes.	Mode	Target state after automatic booting (after a Hardware reset). As a pointer to an unsigned 32-bit value. = 0: "Normal PC Operation State". = 1: "Standalone Basic State". = 2: "Standalone Full State".
Status	Status of automatic booting, see also Chapter 16.7.6 "Automatic Booting – Process in Detail", page 896 . As a pointer to an unsigned 32-bit value. = 0: Finished. = 1: In progress.						
Error	Errors during automatic booting. As a pointer to an unsigned 32-bit value. = 0: No. = 1: Yes.						
Mode	Target state after automatic booting (after a Hardware reset). As a pointer to an unsigned 32-bit value. = 0: "Normal PC Operation State". = 1: "Standalone Basic State". = 2: "Standalone Full State".						
Comments	<ul style="list-style-type: none"> • eth_get_standalone_status requires access rights for the RTC6 Ethernet Board. Otherwise, eth_get_standalone_status has no effect (get_last_error return code RTC6_ACCESS_DENIED). • eth_get_standalone_status has no effect, if the board is not an RTC6 Ethernet Board (get_last_error return code RTC6_TYPE_REJECTED). • Even during automatic booting, access rights can be requested by acquire_rtc. 						
RTC4→RTC6	New command.						
RTC5→RTC6	New command.						
Version info	Available as of DLL 628 , OUT 629 .						
References	–						



Ctrl Command	eth_get_static_ip	
Function	Returns static IP address, subnet mask and gateway address which are saved on the RTC6 Ethernet Board.	
Call	Result = eth_get_static_ip(&Ip, &NetMask, &Gateway)	
Result	Error code. As an unsigned 32-bit value. 0: Success: OK. 1: Error. Not an RTC6 Ethernet Board.	
Parameters	None.	
Returned parameter values	Ip Static IP address in Big Endian byte order. As a pointer to an unsigned 32-bit value. NetMask Sub net mask in Big Endian byte order. As a pointer to an unsigned 32-bit value. Gateway Address of the gateway in Big Endian byte order. As a pointer to an unsigned 32-bit value. All values are each 0, if the board is not an RTC6 Ethernet Board or a static IP address has not been programmed yet.	
Comments	<ul style="list-style-type: none"> To convert from Big Endian byte order to the usual dotted decimal notation, eth_convert_ip_to_string can be used. See also Chapter 16.5.1 "Notes on Working with IP Addresses", page 886. The values can also be queried by eth_get_card_info. 	
RTC4→RTC6	New command.	
RTC5→RTC6	New command.	
Version info	Available as of DLL 606, OUT 606, RBF 611.	
References	eth_convert_ip_to_string , eth_get_card_info , eth_set_static_ip	



Ctrl Command	eth_max_card
Function	Returns the highest index in the RTC6 board management, where an RTC6 Ethernet Board is entered.
Call	<code>Result = eth_max_card()</code>
Parameters	None.
Result	<p>Highest index of an RTC6 Ethernet Board in the RTC6 board management. As an unsigned 32-bit value.</p> <p>0: No RTC6 Ethernet Board recorded in the RTC6 board management.</p> <p>n: Highest index in the RTC6 board management, where an RTC6 Ethernet Board is entered.</p>
Comments	<ul style="list-style-type: none"> • eth_max_card is not available as a multi-board command. • eth_max_card does not return the total number of RTC6 Ethernet Boards in the RTC6 board management. For this purpose, eth_count_cards is available. • With eth_max_card, all RTC6 Ethernet Boards can be removed from the RTC6 board management by a simple loop: <pre>while (eth_max_card()) { release_rtc(eth_max_card()); eth_remove_card(eth_max_card()); }</pre> • See also Chapter 16.5.3 "About the RTC6 Board Management", page 887.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 606, OUT 606, RBF 611.
References	eth_count_cards , eth_remove_card

Ctrl Command	eth_remove_card
Function	Deletes the RTC6 Ethernet Board entry from the RTC6 board management at the specified index.
Call	Result = eth_remove_card(CardNo)
Parameters	CardNo Index of the RTC6 Ethernet Board in the RTC6 board management. As an unsigned 32-bit value.
Result	<p>Error code. As a signed 32-bit value.</p> <p>-2 Error: the entry cannot be deleted. At this index an RTC6 Ethernet Board is entered which is still acquired yet.</p> <p>-1 Error: the entry cannot be deleted. At this index an RTC6 PCIe Board is entered.</p> <p>0 Error: the entry cannot be deleted. At this index "No card" is entered or CardNo is invalid (> 255).</p> <p>n (= CardNo) Success: the entry has been deleted.</p>
Comments	<ul style="list-style-type: none"> • eth_remove_card is not available as a multi-board command. • With all errors, the get_last_error return code RTC6_PARAM_ERROR is generated. • Entries for "No card"s and RTC6 PCIe Boards cannot be deleted. • Before deleting an RTC6 Ethernet Board entry, an acquired board must be explicitly released by release_rtc. • With CardNo = 0, the RTC6 Ethernet Board entry at index (rtc6_count_cards + eth_count_cards) is deleted (see also continuous card numbering without gaps using eth_assign_card or eth_assign_card_ip and CardNo = 0). CardNo = 0 and freely chosen assignment of RTC6 Ethernet Boards to indexes should not be concurrently used. • All RTC6 Ethernet Board entries can be deleted from the RTC6 board management by a simple loop (provided that all RTC6 Ethernet Boards are <i>not</i> acquired): <pre>while (eth_remove_card(0) > 0);</pre> • See also Chapter 16.5.3 "About the RTC6 Board Management", page 887.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 606, OUT 606, RBF 611.
References	eth_assign_card , eth_assign_card_ip , eth_count_cards , release_rtc , rtc6_count_cards



Ctrl Command	eth_search_cards				
Function	Executes a board search in form of a broadcast and returns the number of RTC6 Ethernet Boards which have answered in the specified address range.				
Call	<code>Result = eth_search_cards(Ip, Netmask)</code>				
Parameters	<table> <tr> <td>Ip</td> <td>IP-address in Big Endian byte order. As an unsigned 32-bit value.</td> </tr> <tr> <td>Netmask</td> <td>Subnet mask in Big Endian byte order. As an unsigned 32-bit value.</td> </tr> </table>	Ip	IP-address in Big Endian byte order. As an unsigned 32-bit value.	Netmask	Subnet mask in Big Endian byte order. As an unsigned 32-bit value.
Ip	IP-address in Big Endian byte order. As an unsigned 32-bit value.				
Netmask	Subnet mask in Big Endian byte order. As an unsigned 32-bit value.				
Result	Number of RTC6 Ethernet Boards in the specified address range which have answered (within TimeOut). As an unsigned 32-bit value.				
Comments	<ul style="list-style-type: none"> • eth_search_cards is not available as a multi-board command. • To convert from the usual dotted decimal notation to Big Endian byte order, eth_convert_string_to_ip can be used. See also Chapter 16.5.1 "Notes on Working with IP Addresses", page 886. • The timeout value can be set with eth_set_search_cards_timeout. • A broadcast only reaches reliably addresses within the specified network segment. For other use cases a board search by IP scan (eth_search_cards_range) can be carried out. • The number of found RTC6 Ethernet Boards can also be queried by eth_found_cards at a later time. 				
RTC4→RTC6	New command.				
RTC5→RTC6	New command.				
Version info	Available as of DLL 606, OUT 606, RBF 611.				
References	eth_convert_string_to_ip , eth_found_cards , eth_search_cards_range , eth_set_search_cards_timeout				



Ctrl Command	eth_search_cards_range
Function	Executes a search within a precisely specified IP address range and returns the number of RTC6 Ethernet Boards which have answered.
Call	Result = eth_search_cards_range(StartIP, EndIp)
Parameters	<p>StartIP Start IP-address in Big Endian byte order. As an unsigned 32-bit value.</p> <p>EndIp End IP-address in Big Endian byte order. As an unsigned 32-bit value.</p>
Result	Number of RTC6 Ethernet Boards which have answered (within TimeOut). As an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> • eth_search_cards_range is not available as a multi-board command. • To convert from the usual dotted decimal notation to Big Endian byte order, eth_convert_string_to_ip can be used. See also Chapter 16.5.1 "Notes on Working with IP Addresses", page 886. • eth_search_cards_range executes the search by sending UDP packets to each IP address within the specified address range. It reliably covers the specified address range (compare to broadcast with eth_search_cards). • The TimeOut value can be set with eth_set_search_cards_timeout. • The number of found RTC6 Ethernet Boards can also be queried by eth_found_cards at a later time.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 606, OUT 606, RBF 611.
References	eth_convert_string_to_ip , eth_found_cards , eth_search_cards , eth_set_search_cards_timeout

Ctrl Command	eth_set_com_timeouts												
Function	Sets timing information (for Ethernet communication) for an RTC6 Ethernet Board that is entered in the RTC6 board management (and needs to be acquired, if <code>KeepAlive</code> and <code>KeepInterval</code> are to be changed).												
Call	<code>eth_set_com_timeouts(AcquireTimeout, AcquireMaxRetries, SendRecvTimeout, SendRecvMaxRetries, KeepAlive, KeepInterval)</code>												
Parameters	<table> <tr> <td>AcquireTimeout</td> <td>As an unsigned 32-bit value. With 0 the parameter is not changed.</td></tr> <tr> <td>AcquireMaxRetries</td> <td>See <code>AcquireTimeout</code>.</td></tr> <tr> <td>SendRecvTimeout</td> <td>See <code>AcquireTimeout</code>.</td></tr> <tr> <td>SendRecvMaxRetries</td> <td>See <code>AcquireTimeout</code>.</td></tr> <tr> <td>KeepAlive</td> <td>See <code>AcquireTimeout</code>.</td></tr> <tr> <td>KeepInterval</td> <td>See <code>AcquireTimeout</code>.</td></tr> </table>	AcquireTimeout	As an unsigned 32-bit value. With 0 the parameter is not changed.	AcquireMaxRetries	See <code>AcquireTimeout</code> .	SendRecvTimeout	See <code>AcquireTimeout</code> .	SendRecvMaxRetries	See <code>AcquireTimeout</code> .	KeepAlive	See <code>AcquireTimeout</code> .	KeepInterval	See <code>AcquireTimeout</code> .
AcquireTimeout	As an unsigned 32-bit value. With 0 the parameter is not changed.												
AcquireMaxRetries	See <code>AcquireTimeout</code> .												
SendRecvTimeout	See <code>AcquireTimeout</code> .												
SendRecvMaxRetries	See <code>AcquireTimeout</code> .												
KeepAlive	See <code>AcquireTimeout</code> .												
KeepInterval	See <code>AcquireTimeout</code> .												
Result	None.												
Comments	<ul style="list-style-type: none"> See also Chapter 16.5.3 "About the RTC6 Board Management", page 887. The settings can be queried by eth_get_com_timeouts. All time specifications in μs. As a rule, it is not required to change these parameter values. In case of timing problems contact SCANLAB in order to clarify your special local characteristics. AcquireTimeout, AcquireMaxRetries, SendRecvTimeout and SendRecvMaxRetries refer only to the RTC6 DLL. Therefore, they can be set even without granted access right to an RTC6 Ethernet Board. KeepAlive and KeepInterval are not set, if for the addressed RTC6 Ethernet Board <ul style="list-style-type: none"> there is no access right granted (get_last_error return code <code>RTC6_ACCESS_DENIED</code> and <code>RTC6_ETH_ERROR</code>) there is a granted access right, but no connection is established (get_last_error return code <code>RTC6_ETH_ERROR</code>) \geq DLL 619: No get_last_error return code is set with <code>KeepAlive = KeepInterval = 0</code> because a connection to an RTC6 Ethernet Board is not necessary. eth_set_com_timeouts_auto sets an automatic mechanism, see there. 												
RTC4→RTC6	New command.												
RTC5→RTC6	New command.												
Version info	Available as of DLL 606, OUT 606, RBF 611. Last change DLL 619: No longer an error message, if there is no RTC6 Ethernet Board connection.												
References	eth_get_com_timeouts , eth_set_com_timeouts_auto												

Ctrl Command	eth_set_com_timeouts_auto
Function	Sets timing information (for Ethernet communication) for an RTC6 Ethernet Board that is entered in the RTC6 board management.
Call	<code>eth_set_com_timeouts_auto(InitialTimeout, MaxTimeout, Multiplier, Mode)</code>
Parameters	InitialTimeout The timeout to start with. In ms. As a 64-bit IEEE floating point value.
	MaxTimeout Maximum timeout (= sum of the timeouts of the individual attempts). After MaxTimeout has elapsed, the network connection is considered to be faulty (udp_recv_timeout error, see eth_get_last_error). In ms. As a 64-bit IEEE floating point value.
	Multiplier Multiplication factor by which the timeout is increased for each attempt. As a 64-bit IEEE floating point value.
	Mode Reserved. As an unsigned 32-bit value.
Result	None.
Comments	<ul style="list-style-type: none"> See also Chapter 16.5.3 "About the RTC6 Board Management", page 887. Each control command and list command is sent to the RTC6 Ethernet Board in the form of network packets. For each packet sent, the response packet must be received within a certain timeout. If this timeout is exceeded, the process is repeated. To avoid overload, the next timeout value is increased by the factor <code>Multiplier</code> with each further attempt. The settings can be queried by eth_get_com_timeouts_auto. <code>eth_set_com_timeouts_auto</code> configures an automatic mechanism to determine appropriate timeout values. All time specifications in ms, not μs. If 0 is specified for a parameter, its current value is not changed. Default parameter values (after <code>eth_assign_card</code> and <code>eth_assign_card_ip</code>): see eth_get_com_timeouts_auto. The parameter <code>Mode</code> is reserved for future extensions. <code>eth_set_com_timeouts_auto</code> affects solely the RTC6 DLL. There is no need for a connection to the RTC6 Ethernet Board. <code>eth_set_com_timeouts</code> sets a static mechanism.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 620, OUT 620.
References	eth_get_com_timeouts_auto , eth_set_com_timeouts



Ctrl Command	eth_set_high_performance_mode
Function	Switches on (off) the “High Performance Mode” for an RTC6 Ethernet Board.
Prerequisite	Minimum requirement: RTC6 Software Package V1.14.1 and BIOS-ETH 35.
Call	ErrorCode = eth_set_high_performance_mode(Mode)
Parameters	<p>Mode</p> <p>Mode.</p> <p>As an unsigned 32-bit value.</p> <p>= 0: Switches the “High Performance Mode” off.</p> <p>= 1: Switches the “High Performance Mode” on.</p>
Result	<p>Error code.</p> <p>As an unsigned 32-bit value.</p> <p>0: Success: OK.</p> <p>1: Error: Mode not allowed. get_last_error return code RTC6_PARAM_ERROR.</p> <p>2: Error: BIOS not supported (< BIOS-ETH 35). get_last_error-return code RTC6_UNSUPPORTED_BIOS.</p> <p>3: Error: No access to the RTC6 Ethernet Board. get_last_error return code RTC6_ACCESS_DENIED.</p> <p>4: Error: Not an RTC6 Ethernet Board. get_last_error return code RTC6_TYPE_REJECTED.</p>
Comments	<ul style="list-style-type: none"> In “High Performance Mode”, the network load can increase significantly. <ul style="list-style-type: none"> For compensation, eth_set_high_performance_mode calls internally eth_set_com_timeouts_auto(0.75, 1000.0, 1.3, 1). If eth_set_com_timeouts has been previously called with larger values, they are not be changed. Default setting after eth_assign_card and eth_assign_card_ip: <ul style="list-style-type: none"> Mode = 0 load_program_file does not reset the setting. See also Chapter 16.8 ““High Performance Mode””, page 897.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 633.
References	–

Ctrl Command	eth_set_port_numbers								
Function	Saves the UDP ports for board searches (UDPsearch) and exclusive connections (UDPexcl) as well as the TCP port into the Flash memory of the RTC6 Ethernet Board.								
Call	Result = eth_set_port_numbers(UDPsearch , UDPexcl , TCP)								
Parameters	<table> <tr> <td>UDPsearch</td> <td>UDP port for board searches. As an unsigned 32-bit value. Allowed value range: [0...65,535]. Out-of-range values are clipped.</td> </tr> <tr> <td>UDPexcl</td> <td>UDP port for exclusive connections. As an unsigned 32-bit value. Allowed value range: [0...65,535]. Out-of-range values are clipped.</td> </tr> <tr> <td>TCP</td> <td>TCP port for exclusive connections. As an unsigned 32-bit value. Allowed value range: [0...65,535]. Out-of-range values are clipped.</td> </tr> </table>	UDPsearch	UDP port for board searches. As an unsigned 32-bit value. Allowed value range: [0...65,535]. Out-of-range values are clipped.	UDPexcl	UDP port for exclusive connections. As an unsigned 32-bit value. Allowed value range: [0...65,535]. Out-of-range values are clipped.	TCP	TCP port for exclusive connections. As an unsigned 32-bit value. Allowed value range: [0...65,535]. Out-of-range values are clipped.		
UDPsearch	UDP port for board searches. As an unsigned 32-bit value. Allowed value range: [0...65,535]. Out-of-range values are clipped.								
UDPexcl	UDP port for exclusive connections. As an unsigned 32-bit value. Allowed value range: [0...65,535]. Out-of-range values are clipped.								
TCP	TCP port for exclusive connections. As an unsigned 32-bit value. Allowed value range: [0...65,535]. Out-of-range values are clipped.								
Result	<p>Error code. As an unsigned 32-bit value.</p> <table> <tr> <td>0:</td> <td>Success: OK.</td> </tr> <tr> <td>1:</td> <td>Error: No access to the RTC6 Ethernet Board. get_last_error return code RTC6_ACCESS_DENIED.</td> </tr> <tr> <td>2:</td> <td>Error: No RTC6 Ethernet Board. get_last_error return code RTC6_TYPE_REJECTED.</td> </tr> <tr> <td>3:</td> <td>Error: Programming the Flash memory is not possible. The RTC6 Ethernet Board is possibly busy. get_last_error return code RTC6_BUSY or RTC6_FLASH_ERROR.</td> </tr> </table>	0:	Success: OK.	1:	Error: No access to the RTC6 Ethernet Board. get_last_error return code RTC6_ACCESS_DENIED .	2:	Error: No RTC6 Ethernet Board. get_last_error return code RTC6_TYPE_REJECTED .	3:	Error: Programming the Flash memory is not possible. The RTC6 Ethernet Board is possibly busy. get_last_error return code RTC6_BUSY or RTC6_FLASH_ERROR .
0:	Success: OK.								
1:	Error: No access to the RTC6 Ethernet Board. get_last_error return code RTC6_ACCESS_DENIED .								
2:	Error: No RTC6 Ethernet Board. get_last_error return code RTC6_TYPE_REJECTED .								
3:	Error: Programming the Flash memory is not possible. The RTC6 Ethernet Board is possibly busy. get_last_error return code RTC6_BUSY or RTC6_FLASH_ERROR .								
Comments	<ul style="list-style-type: none"> Prerequisites for eth_set_port_numbers: <ul style="list-style-type: none"> No list must currently be executed on the RTC6 Ethernet Board. The RTC6 Ethernet Board must be entered in the RTC6 board management. The RTC6 Ethernet Board must have been acquired. If 0 is specified for a parameter, it is not overwritten. Saved parameters are not used until the RTC6 Ethernet Board has been restarted. 								
RTC4→RTC6	New command.								
RTC5→RTC6	New command.								
Version info	Available as of DLL 606, OUT 606, RBF 611.								
References	eth_get_port_numbers								



Ctrl Command	eth_set_search_cards_timeout
Function	Sets a timeout value.
Call	<code>eth_set_search_cards_timeout(TimeOut)</code>
Parameters	TimeOut The timeout value. In μ s. As an unsigned 32-bit value.
Result	None.
Comments	<ul style="list-style-type: none">• <code>eth_set_search_cards_timeout</code> is not available as a multi-board command (<code>n_</code>).• The default timeout value is 5 ms.• The timeout value is relevant for <code>eth_search_cards</code> and <code>eth_search_cards_range</code>.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 606, OUT 606, RBF 611.
References	eth_search_cards , eth_search_cards_range



Ctrl Command	eth_set_static_ip								
Function	Writes a static IP address, a subnet mask and also a gateway address onto the RTC6 Ethernet Board.								
Call	<code>Result = eth_set_static_ip(Ip, NetMask, Gateway)</code>								
Parameters	<table> <tr> <td>Ip</td> <td>Static IP address in Big Endian byte order. As an unsigned 32-bit value.</td> </tr> <tr> <td>NetMask</td> <td>Subnet mask in Big Endian byte order. As an unsigned 32-bit value.</td> </tr> <tr> <td>Gateway</td> <td>Address of the gateway in Big Endian byte order. As an unsigned 32-bit value.</td> </tr> </table>	Ip	Static IP address in Big Endian byte order. As an unsigned 32-bit value.	NetMask	Subnet mask in Big Endian byte order. As an unsigned 32-bit value.	Gateway	Address of the gateway in Big Endian byte order. As an unsigned 32-bit value.		
Ip	Static IP address in Big Endian byte order. As an unsigned 32-bit value.								
NetMask	Subnet mask in Big Endian byte order. As an unsigned 32-bit value.								
Gateway	Address of the gateway in Big Endian byte order. As an unsigned 32-bit value.								
Result	<p>Error code. As an unsigned 32-bit value.</p> <table> <tr> <td>0:</td> <td>Success: OK.</td> </tr> <tr> <td>1:</td> <td>Error: No access to the RTC6 Ethernet Board. get_last_error return code <code>RTC6_ACCESS_DENIED</code>.</td> </tr> <tr> <td>2:</td> <td>Error: No RTC6 Ethernet Board. get_last_error return code <code>RTC6_TYPE_REJECTED</code>.</td> </tr> <tr> <td>3:</td> <td>Error: Programming the Flash memory is not possible. The RTC6 Ethernet Board is possibly busy. get_last_error return code <code>RTC6_BUSY</code> or <code>RTC6_FLASH_ERROR</code>.</td> </tr> </table>	0:	Success: OK.	1:	Error: No access to the RTC6 Ethernet Board. get_last_error return code <code>RTC6_ACCESS_DENIED</code> .	2:	Error: No RTC6 Ethernet Board. get_last_error return code <code>RTC6_TYPE_REJECTED</code> .	3:	Error: Programming the Flash memory is not possible. The RTC6 Ethernet Board is possibly busy. get_last_error return code <code>RTC6_BUSY</code> or <code>RTC6_FLASH_ERROR</code> .
0:	Success: OK.								
1:	Error: No access to the RTC6 Ethernet Board. get_last_error return code <code>RTC6_ACCESS_DENIED</code> .								
2:	Error: No RTC6 Ethernet Board. get_last_error return code <code>RTC6_TYPE_REJECTED</code> .								
3:	Error: Programming the Flash memory is not possible. The RTC6 Ethernet Board is possibly busy. get_last_error return code <code>RTC6_BUSY</code> or <code>RTC6_FLASH_ERROR</code> .								
Comments	<ul style="list-style-type: none"> Prerequisites for <code>eth_set_static_ip</code>: <ul style="list-style-type: none"> No list must currently be executed on the RTC6 Ethernet Board. The RTC6 Ethernet Board must be entered in the RTC6 board management. The RTC6 Ethernet Board must have been acquired. If no gateway is to be used, then <code>Gateway</code> must be set to 0. To convert from usual dotted decimal notation to Big Endian byte order, eth_convert_string_to_ip can be used. See also Chapter 16.5.1 "Notes on Working with IP Addresses", page 886. 								
RTC4→RTC6	New command.								
RTC5→RTC6	New command.								
Version info	Available as of DLL 606, OUT 606, RBF 611.								
References	eth_convert_string_to_ip , eth_get_static_ip								



Ctrl Command	execute_at_pointer
Function	Starts list execution ("List 1" or "List 2") at the specified address in the RTC6 list memory.
Call	<code>execute_at_pointer(Pos)</code>
Parameters	<p>Pos Absolute address of the first list command to be executed. As an unsigned 32-bit value. Allowed value range: [0...(2²³-1)].</p>
Comments	<ul style="list-style-type: none"> • <code>execute_at_pointer</code> essentially functions like <code>execute_list_pos</code> (see 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 <code>Pos</code> \geq <code>Mem1 + Mem2</code> (see <code>config_list</code>), <code>Pos</code> is set to 0.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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>
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>
Comments	<ul style="list-style-type: none"> • <code>execute_list</code> is synonymous with <code>execute_list_pos</code> with <code>Pos = 0</code>. • <code>execute_list_1</code> and <code>execute_list_2</code> (with no parameters) can be used alternatively.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	execute_list

Ctrl Command	execute_list_2
Function	See execute_list .
Call	<code>execute_list_2()</code>
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 value range: [0...(2²³-1)].</p>
Comments	<ul style="list-style-type: none"> • execute_list_pos is not executed (get_error return value = RTC6_BUSY), if: <ul style="list-style-type: none"> – the BUSY list execution status is set – a list has been paused by set_wait (PAUSED list execution status set) • If the INTERNAL-BUSY list execution status is set, execute_list_pos is only executed with a delay (after INTERNAL-BUSY list execution status 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 Chapter 6.4 "List Handling", page 104). • Programs in the protected list memory area "List 3" cannot be directly executed by execute_list_pos. They can only be called from a list ("List 1" or "List 2") as a subroutine. Alternatively, the corresponding area can be assigned by config_list to "List 1" or "List 2". • Uneven ListNo 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 (Mem2 = 0, see config_list) then "List 1" is opened. • If Pos is specified as being larger than the memory area of the respective list (Pos > Mem1 or Pos > Mem2), then Pos 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 read_status). The BUSY list execution status-List Execution Status (see get_status) is set. • Execution stops when a set_end_of_list is encountered. If the end of a list area is reached without encountering a set_end_of_list, 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 set_end_of_list has been encountered and an auto_change_pos or start_loop has been previously called. For both lists to be treated as a single list, you must set the configuration appropriately: for example, config_list(Mem1+Mem2, 0).



Ctrl Command	execute_list_pos
Comments (cont'd)	<ul style="list-style-type: none"> If a home jump (defined with <code>home_position</code> or <code>home_position_xyz</code>) has been executed by <code>set_end_of_list</code>, then <code>execute_list_pos</code> leads to a corresponding home return (the INTERNAL-BUSY list execution status is set while the home return is executed). <code>execute_list_pos</code> triggers a flush of the buffered list input (see Chapter 6.4.1 "Loading Lists", page 104), even if the start has been 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>.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 and subsequently executes a jump to the defined new output position.
Restriction	If the Option Processing-on-the-fly is not enabled, fly_return only executes the jump to the specified new output position.
Call	<code>fly_return(X, Y)</code>
Parameters	<p>X Absolute x coordinate of the new output position. In bits. As a signed 32-bit value. Allowed value range: [-524,288...+524,287]. Out-of-range values are clipped to the boundary values.</p> <p>Y Like X (analogously).</p>
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 has been 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 113). See also Chapter 8.6 "Processing-on-the-fly", page 241.
RTC4→RTC6	Unchanged functionality. In addition: increased value range, and "AbsCall", see above. In RTC4 Compatibility Mode , the RTC6 multiplies the values specified for X and Y by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_fly_x , set_fly_y , set_fly_rot , set_fly_x_pos , set_fly_y_pos , set_fly_rot_pos

Variable List Command	fly_return_1_axis
Function	“Fly Extension” Command: Deactivates 1 Axis of a Processing-on-the-fly application and subsequently carries-out a jump to the specified new output position.
Restriction	If the Option Processing-on-the-fly is not enabled, then fly_return_1_axis only carries-out the jump to the specified new output position.
Call	<code>fly_return_1_axis(Axis, RetPos1)</code>
Parameters	<p>Axis Axis from Table 3, page 258. As an unsigned 32-bit value.</p> <p>RetPos1 Absolute axis coordinate of the new output position. In bits. As a signed 32-bit value. Allowed value range: [-524,288...+524,287]. Out-of-range values are clipped to the boundary values.</p>
Comments	<ul style="list-style-type: none"> Being an “Fly Extension” Command, fly_return_1_axis must not be used mixed with “Classic” Processing-on-the-fly commands (see Footnote, page 241). See Chapter 8.6 “Processing-on-the-fly”, page 241 and Section “Fly Extension” Commands, page 258. Axis needs to be a linear axis (1, 2 or 3). fly_return_2_axes is to be used to deactivate a Rotary axis. fly_return_1_axis only deactivates 1 Axis of a Processing-on-the-fly application, if it has been activated by an “Fly Extension” Command. If no Processing-on-the-fly functionality is active at the specified Axis, only the jump is carried out. With an unallowed parameter value, fly_return_1_axis is replaced by a list_nop (get_last_error return code RTC6_PARAM_ERROR). See also comments on fly_return.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 617, OUT 617, RBF 623.
References	fly_return_2_axes , fly_return_3_axes

Variable List Command	fly_return_2_axes
Function	"Fly Extension" Command: Deactivates 2 Axes of a Processing-on-the-fly application and subsequently carries-out a jump to the specified new output position.
Restriction	If the Option Processing-on-the-fly is not enabled, then fly_return_2_axes only carries-out the jump to the specified new output position.
Call	<code>fly_return_2_axes(Axis1, RetPos1, Axis2, RetPos2)</code>
Parameters	<p>Axis1 Axis from Table 3, page 258. As an unsigned 32-bit value.</p> <p>RetPos1 Absolute axis coordinate of the new output position. In bits. As a signed 32-bit value. Allowed value range: [-524,288...+524,287]. Out-of-range values are clipped to the boundary values.</p> <p>Axis2 Like Axis1.</p> <p>RetPos2 Like RetPos1.</p>
Comments	<ul style="list-style-type: none"> Being an "Fly Extension" Command, fly_return_2_axes must not be used mixed with "Classic" Processing-on-the-fly commands (see Footnote, page 241). See Chapter 8.6 "Processing-on-the-fly", page 241 and Section ""Fly Extension" Commands", page 258. Axis1 and Axis2 need to be 2 different linear axes (1, 2 or 3) or both the Rotary axis (4). In the latter case RetPos1/RetPos2 mean the return coordinates of the Axis 1/2. fly_return_2_axes only deactivates Axes of a Processing-on-the-fly application, if these have been activated by an "Fly Extension" Command. If no Processing-on-the-fly functionality is active at one of the specified Axes, only the jump is carried out. With an unallowed parameter value, fly_return_2_axes is replaced by a list_nop (get_last_error return code <code>RTC6_PARAM_ERROR</code>). See also comments on fly_return. The following command calls are executed in the same way: <ul style="list-style-type: none"> <code>fly_return_2_axes(1, X, 2, Y) = fly_return(X, Y)</code> <code>fly_return_2_axes(4, X, 4, Y) = fly_return(X, Y)</code>
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 617, OUT 617, RBF 623.
References	fly_return_1_axis , fly_return_3_axes

Variable List Command	<code>fly_return_3_axes</code>
Function	"Fly Extension" Command: Deactivates all 3 Axes of a Processing-on-the-fly application and subsequently carries-out a jump to the specified new output position.
Restriction	If the Option Processing-on-the-fly is not enabled, then <code>fly_return_3_axes</code> terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	<code>fly_return_3_axes(RetPosX, RetPosY, RetPosZ)</code>
Parameters	<p>RetPosX Absolute axis coordinate of the new output position. In bits. As a signed 32-bit value. Allowed value range: [-524,288...+524,287]. Out-of-range values are clipped to the boundary values.</p> <p>RetPosY Like RetPosX.</p> <p>RetPosZ Like RetPosX.</p>
Comments	<ul style="list-style-type: none"> Being an "Fly Extension" Command, <code>fly_return_3_axes</code> must not be used mixed with "Classic" Processing-on-the-fly commands (see Footnote, page 241). See Chapter 8.6 "Processing-on-the-fly", page 241 and Section "Fly Extension" Commands, page 258. <code>fly_return_3_axes</code> only deactivates Axes of a Processing-on-the-fly application, if these have been activated by an "Fly Extension" Command. If no Processing-on-the-fly functionality is active at one of the specified Axes, only the jump is carried out. With an unallowed parameter value, <code>fly_return_3_axes</code> is replaced by a list_nop (get_last_error return code <code>RTC6_PARAM_ERROR</code>). See also comments on fly_return. The following command calls are executed in the same way: <ul style="list-style-type: none"> <code>fly_return_3_axes(X, Y, Z) = fly_return_z(X, Y, Z)</code>
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 617, OUT 617, RBF 623.
References	fly_return_1_axis , fly_return_2_axes



Normal List Command	fly_return_z						
Function	Deactivates the previously set Processing-on-the-fly correction. Subsequently executes a jump to the defined position.						
Restriction	If the Option Processing-on-the-fly is not enabled, fly_return_z only executes the jump to the defined new output position.						
Call	<code>fly_return_z(X, Y, Z)</code>						
Parameters	<table> <tr> <td>X</td> <td>Absolute x coordinate of the new output position. In bits. As a signed 32-bit value. Allowed value range: [-524,288...+524,287]. Out-of-range values are clipped to the boundary values.</td> </tr> <tr> <td>Y</td> <td>Like X (analogously).</td> </tr> <tr> <td>Z</td> <td>Like X (analogously).</td> </tr> </table>	X	Absolute x coordinate of the new output position. In bits. As a signed 32-bit value. Allowed value range: [-524,288...+524,287]. Out-of-range values are clipped to the boundary values.	Y	Like X (analogously).	Z	Like X (analogously).
X	Absolute x coordinate of the new output position. In bits. As a signed 32-bit value. Allowed value range: [-524,288...+524,287]. Out-of-range values are clipped to the boundary values.						
Y	Like X (analogously).						
Z	Like X (analogously).						
Comments	<ul style="list-style-type: none"> Like fly_return, however, with additional Z output position. 						
RTC4→RTC6	<p>New command.</p> <p>In RTC4 Compatibility Mode, the RTC6 multiplies the specified value for X, Y and Z by 16. The allowed value range decreases accordingly.</p>						
RTC5→RTC6	<p>Unchanged functionality.</p> <p>In RTC5 Compatibility Mode, the RTC6 multiplies the specified value for Z by 16. The allowed value range decreases accordingly.</p>						
Version info	Available as of DLL 600, OUT 600, RBF 600.						
References	set_fly_z						

Ctrl Command	free_RTC6_dll
Function	Frees up all resources allocated by the RTC6 DLL for a user program.
Call	<code>free_RTC6_dll()</code>
Comments	<ul style="list-style-type: none"> • RTC6 DLL-allocated resources particularly include memory area in the RTC6 DLL allocated for the RTC6 board management (which is created by init_RTC6_dll). free_RTC6_dll deletes the RTC6 board management of the RTC6 DLL. Afterward, the user program has no access to boards (get_last_error return code RTC6_ACCESS_DENIED). • The calling of free_RTC6_dll is not absolutely necessary, because RTC6 DLL-assigned resources are automatically freed up when the user program terminates and the RTC6 DLL is thereby unloaded by Microsoft Windows. However, some user program development environments (in debug mode) issue "memory leaks detected" warnings even though the RTC6 DLL is unloaded. The calling of free_RTC6_dll eliminates this annoyance. • free_RTC6_dll is not available as a multi-board command.
RTC4→RTC6	New command.
RTC5→RTC6	New command. The functionalities of free_RTC6_dll and the RTC5 command free_RTC5_dll are identical.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	init_RTC6_dll , release_RTC



Ctrl Command	get_auto_cal
Function	Returns the type of ASC hardware integrated in the attached scan system previously detected by auto_cal .
Call	ASCType = get_auto_cal(HeadNo)
Parameters	HeadNo Number of the scan head connector. As an unsigned 32-bit value. Allowed values: = 1: First scan head connector. = 2: Second scan head connector.
Result	ASC hardware type. As an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> If the ASC hardware type has been previously detected by auto_cal, then get_auto_cal returns the same value as auto_cal(Command = 4), see comments there. If the ASC hardware type has <i>not</i> been previously detected by auto_cal, then get_auto_cal returns the value 255 (initialized value).
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	auto_cal

Ctrl Command	get_bios_version
Function	Returns the BIOS version number of the RTC6 Board.
Call	<code>BiosVersion = get_bios_version()</code>
Result	BCD-coded BIOS version number. As an unsigned 32-bit value. Example: <code>BiosVersion = 33 = 0x21</code> means BIOS version 21.
Parameters	None.
Comments	• get_bios_version returns reliable results only as of BIOS version 21, otherwise 0.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 606, OUT 606, RBF 611.
References	get_dll_version , get_hex_version , get_RTC_version

Ctrl Command	get_card_type
Function	Returns the RTC6 board type.
Call	<code>Result = get_card_type()</code>
Result	Board type. As an unsigned 32-bit value. 0: "No card". 1: RTC6 PCIe Board. 2: RTC6 Ethernet Board.
Parameters	None.
Comments	• See also Chapter 16.5.3 "About the RTC6 Board Management", page 887 .
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 606, OUT 606, RBF 611.
References	eth_assign_card , eth_assign_card_ip

Ctrl Command	get_char_pointer
Function	Returns the absolute start address of an indexed character.
Call	CharPointer = get_char_pointer(Char)
Parameters	Char Index of the indexed character. As an unsigned 32-bit value. Allowed value range: [0...1023].
Result	Absolute start address. As an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> • get_char_pointer 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 list memory area depends on whether the character has been loaded into the protected list memory area "List 3" or an unprotected subroutine has been only subsequently referenced. • If <code>Index > 1023</code> or if no character has been referenced with the specified index, then get_char_pointer returns the value "-1" (for example, $2^{32}-1$). • get_char_pointer 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, for example, 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 list_call_cond is ignored.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	get_sub_pointer, get_text_table_pointer

Ctrl Command	get_config_list
Function	Passes the parameters of the current list memory configuration (<code>Mem1</code> , <code>Mem2</code>) to the RTC6 board management of the RTC6 DLL and initializes it as if config_list would have been called.
Call	<code>get_config_list()</code>
Result	–
Comments	<ul style="list-style-type: none"> The get_config_list command is useful when a board changes “ownership” and the new RTC6 board management is not aware of the memory configuration (at the start of each user program, the board and board management each independently initialize <code>Mem1</code> = 4,194,304 and <code>Mem2</code> = 4,194,304; the board by load_program_file and board management when starting the corresponding user program). See also Chapter 6.7.1 “Notes on Board Acquisition by a User Program”, page 127. get_config_list does not return a value to the user program. The user program can, however, read the list-memory configuration data after <code>load_list(ListNo, 0)</code> or <code>set_start_list_pos(ListNo, 0)</code> by using get_list_space. get_config_list is executed regardless of the BUSY list execution status.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	config_list

Ctrl Command	get_counts
Function	Reads the current number of successful External Starts .
Call	<code>Counts = get_counts()</code>
Result	Number of successful External 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. This counter can be reset to 0 by set_control_mode.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_max_counts , set_control_mode , get_startstop_info

Ctrl Command	get_dll_version
Function	Returns the version number of the RTC6 DLL .
Call	<code>DLLVersion = get_dll_version()</code>
Result	Version number. As an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> The RTC6 DLL version numbers are in the range 600...699. get_dll_version is available even without explicit access rights to a specific RTC6 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 Chapter 6.8 "Error Handling", page 129, are neither generated nor altered by get_dll_version.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	get_hex_version , get_RTC_version

Ctrl Command	get_encoder				
Function	Returns the current counts of the two internal encoder counters.				
Call	<code>get_encoder(&Encoder0, &Encoder1)</code>				
Returned parameter values	<table> <tr> <td>Encoder0</td> <td>Current count of encoder counter "Encoder0". As a pointer to a signed 32-bit value.</td> </tr> <tr> <td>Encoder1</td> <td>Current count of encoder counter "Encoder1". As a pointer to a signed 32-bit value.</td> </tr> </table>	Encoder0	Current count of encoder counter "Encoder0". As a pointer to a signed 32-bit value.	Encoder1	Current count of encoder counter "Encoder1". As a pointer to a signed 32-bit value.
Encoder0	Current count of encoder counter "Encoder0". As a pointer to a signed 32-bit value.				
Encoder1	Current count of encoder counter "Encoder1". As a pointer to a signed 32-bit value.				
Comments	<ul style="list-style-type: none"> For usage of get_encoder, see Chapter 8.6 "Processing-on-the-fly", page 241 and Chapter 9.3.3 "Synchronization by Encoder Signals", page 297. If the workpiece movement is registered with an incremental encoder: <ul style="list-style-type: none"> Encoder counter "Encoder0" is triggered by the signals at encoder input port ENCODER X Encoder counter "Encoder1" is triggered by the signals at encoder input port ENCODER Y If an encoder simulation has been started by simulate_encoder, the encoder counters are triggered by an internal periodic 1 MHz clock signal. 				
RTC4→RTC6	Unchanged functionality. In addition: increased value range.				
RTC5→RTC6	Unchanged functionality.				
Version info	Available as of DLL 600, OUT 600, RBF 600.				
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. It corresponds to 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. For the specific errors the error constants should be predefined as mentioned below.</p> <table> <thead> <tr> <th>Bit</th> <th>Error type</th> <th>Error constant</th> <th></th> </tr> </thead> <tbody> <tr> <td>–</td> <td>No error.</td> <td>RTC6_NO_ERROR</td> <td>= 0</td> </tr> <tr> <td>Bit #0 (LSB)</td> <td>= 1: No RTC6 PCIe Board found. This error can only occur with init_rtc6_dll.</td> <td>RTC6_NO_PCIE_CARD_FOUND</td> <td>= 1</td> </tr> <tr> <td>Bit #1</td> <td>= 1: Access denied. This error can occur by init_rtc6_dll, select_rtc, acquire_rtc or any multi-board command.</td> <td>RTC6_ACCESS_DENIED</td> <td>= 2</td> </tr> <tr> <td>Bit #2</td> <td>= 1: Command not forwarded. This error implies an internal, RTC6 board driver error or PCI error, for example, caused by a hardware defect or an incorrect connection.</td> <td>RTC6_SEND_ERROR</td> <td>= 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 RTC6. This error can especially occur in connection with control commands that expect a response, for example, get_head_para.</td> <td>RTC6_TIMEOUT</td> <td>= 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, for example, 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>RTC6_PARAM_ERROR</td> <td>= 16</td> </tr> </tbody> </table>			Bit	Error type	Error constant		–	No error.	RTC6_NO_ERROR	= 0	Bit #0 (LSB)	= 1: No RTC6 PCIe Board found. This error can only occur with init_rtc6_dll .	RTC6_NO_PCIE_CARD_FOUND	= 1	Bit #1	= 1: Access denied. This error can occur by init_rtc6_dll , select_rtc , acquire_rtc or any multi-board command.	RTC6_ACCESS_DENIED	= 2	Bit #2	= 1: Command not forwarded. This error implies an internal, RTC6 board driver error or PCI error, for example, caused by a hardware defect or an incorrect connection.	RTC6_SEND_ERROR	= 4	Bit #3	= 1: No response from board. It is likely that no program has been loaded onto the RTC6. This error can especially occur in connection with control commands that expect a response, for example, get_head_para .	RTC6_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, for example, 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.	RTC6_PARAM_ERROR	= 16
Bit	Error type	Error constant																													
–	No error.	RTC6_NO_ERROR	= 0																												
Bit #0 (LSB)	= 1: No RTC6 PCIe Board found. This error can only occur with init_rtc6_dll .	RTC6_NO_PCIE_CARD_FOUND	= 1																												
Bit #1	= 1: Access denied. This error can occur by init_rtc6_dll , select_rtc , acquire_rtc or any multi-board command.	RTC6_ACCESS_DENIED	= 2																												
Bit #2	= 1: Command not forwarded. This error implies an internal, RTC6 board driver error or PCI error, for example, caused by a hardware defect or an incorrect connection.	RTC6_SEND_ERROR	= 4																												
Bit #3	= 1: No response from board. It is likely that no program has been loaded onto the RTC6. This error can especially occur in connection with control commands that expect a response, for example, get_head_para .	RTC6_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, for example, 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.	RTC6_PARAM_ERROR	= 16																												

Ctrl Command	get_error			
Result (cont'd)	Bit #5 = 1: List processing is (not) active. Examples: for execute_list , if a list is currently being processed; for stop_execution , if no list is currently being processed; for restart_list , if pause_list has not been previously called.	RTC6_BUSY		= 32
	Bit #6 = 1: List command rejected, invalid input pointer. For example, for any list command directly after load_char + list_return : the list command is then not loaded.	RTC6_REJECTED		= 64
	Bit #7 = 1: List command has been converted to a list_nop . For example, set_end_of_list in a protected subroutine.	RTC6_IGNORED		= 128
	Bit #8 = 1: Version error: RTC6 DLL version, RTC6RBF.rbf version and RTC6OUT.out / RTC6ETH.out version are not compatible with each other. See also load_program_file .	RTC6_VERSION_MISMATCH		= 256
	Bit #9 = 1: Verify error: The download verification has detected an incorrect download. See also Chapter 6.8.1 "Download Verification", page 130 .	RTC6_VERIFY_ERROR		= 512
	Bit #10 = 1: For example, an [*]eth[*] command has been sent to an RTC6 PCIe Board.	RTC6_TYPE_REJECTED		= 1024
	Bit #11 = 1: A RTC6 DLL -internal Windows memory request failed.	RTC6_OUT_OF_MEMORY		= 2048
	Bit #12 = 1: Download error. The values have possibly not been saved. May occur with auto_cal and write_hi_pos .	RTC6_FLASH_ERROR		= 4096

Ctrl Command	get_error			
Result (cont'd)	Bit #13	= 1: General Ethernet error. May occur, for example, when trying to switch by select_rtc or acquire_rtc from RTC6 PCIe Board to RTC6 Ethernet Board. Further information is provided by n_get_error on the addressed RTC6 Ethernet Board.	RTC6_ETH_ERROR	= 8192
	Bit #14	Reserved.	—	
	Bit #15	= 1: Unsupported Windows version. May only occur during init_rtc6_dll .	RTC6_UNSUPPORTED_WINDOWS	= 32768
	Bit #16	Reserved.	—	
	Bit #17	= 1: Unsupported BIOS version.	RTC6_UNSUPPORTED_BIOS	= 131072
	Bit #18	Reserved.	—	
		
	Bit #31	Reserved.	—	
Comments	<ul style="list-style-type: none"> For error handling see Chapter 6.8 "Error Handling", page 129. get_error and n_get_error are available even without explicit access rights to a specific RTC6 board. The board-specific error variables LastError and AccError (see Chapter 6.8 "Error Handling", page 129) are neither generated nor altered by get_error. Bit #0 = 1 (error constant: RTC6_NO_PCIE_CARD_FOUND = 1) means that no RTC6 PCIe Board has been found (init_rtc6_dll does not search for RTC6 Ethernet Boards). 			
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_rtc6_dll(); // Searches for all installed RTC6 boards if (Error & RTC6_ACCESS_DENIED) { // at least one board is inaccessible UINT Count = rtc6_count_cards(); // number of boards found for (UINT Num = 1; Num <= Count; Num++) { NoAccess[Num] = n_get_last_error(Num) & RTC6_ACCESS_DENIED; n_reset_error(Num, RTC6_ACCESS_DENIED); } } </pre>			



Ctrl Command	get_error
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 x reference value. As a pointer to a signed 32-bit value. OffsetY y reference value. As a pointer to a signed 32-bit value.
Comments	<ul style="list-style-type: none"> For 2D encoder compensation, see Section "2D Encoder Compensation for xy Positioning Stages", page 248.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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>
Parameters	No Number of the free variable to be queried. As an unsigned 32-bit value. Allowed value range: [0...7]. Only the 3 least significant bits are evaluated.
Result	The value currently stored in the free variable No. As an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> See also Chapter 6.9.1 "Free Variables", page 134.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_free_variable , set_free_variable_list

Ctrl Command	get_galvo_controls
Function	Returns the corresponding control values for given input values.
Restriction	get_galvo_controls can only be executed, if no list is currently being processed (get_last_error return code RTC6_BUSY).
Call	<code>get_galvo_controls(InPtr, OutPtr)</code>
Parameters	<p>InPtr Pointer (data type ULONG_PTR in C and C++, an unsigned 32-bit or unsigned 64-bit value) to an array of five unsigned 32-bit values, where the to-be-outputted settings are specified: X, Y, Z, Defocus, Zoom. Out-of-range values are clipped to the boundary values.</p> <p>OutPtr Pointer (data type ULONG_PTR in C and C++, an unsigned 32-bit or unsigned 64-bit value) to an array of 4 unsigned 32-bit values, where the corresponding control values are to be stored: XA, YA, XB, YB. Value range in each case [-524,288...+524,287].</p>
Returned parameter values	XA, YA, XB, YB denote the control values for the x axis/y axis of scan head A/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 get_galvo_controls (for example, matrix, offset, angle, etc.; also stretch table) are considered as they are set at the moment. Make sure to apply these by <code>at_once > 0</code> before calling get_galvo_controls!. 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. <p><code>select_cor_table(0,0): XA = YA = XB = YB = 0</code></p> <p>2D correction files: inputs Z, Defocus, Zoom are ignored</p> <p><code>select_cor_table(1,0): XA, YA calculated, XB, YB = 0</code></p> <p><code>select_cor_table(0,1): XA, YA = 0, XB, YB calculated</code></p> <p><code>select_cor_table(1,1): XA, YA, XB, YB calculated</code></p> <p>(Standard-)3D correction file: input Zoom is ignored</p> <p><code>select_cor_table(1,0): XA, YA calculated, XB = YB = Zout calculated</code></p> <p><code>select_cor_table(0,1): XA= YA = Zout calculated, XB, YB calculated</code></p> <p><code>select_cor_table(1,1): same as select_cor_table(0,0)</code></p> <p>3D zoom correction file (only for intelliWELD II with zoom axis):</p> <p><code>select_cor_table(1,0): XA, YA calculated, XB = Zout, YB = ZoomOut calculated</code></p> <p><code>select_cor_table(0,1): XA= Zout, YA = ZoomOut calculated, XB, YB calculated</code></p> <ul style="list-style-type: none"> • The return values are 0, if a get_last_error return code RTC6_BUSY has been generated.



Ctrl Command	get_galvo_controls
RTC4→RTC6	<p>New command.</p> <p>In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for x, y, z and Defocus by 16. The allowed value range decreases accordingly.</p> <p>Even in RTC4 Compatibility Mode, all returned values are in the RTC6 20-bit range.</p>
RTC5→RTC6	<p>Unchanged functionality.</p> <p>In RTC5 Compatibility Mode, the RTC6 multiplies the specified values for z and Defocus by 16. The allowed value range decreases accordingly.</p> <p>Even in RTC5 Compatibility Mode, all returned values are in the RTC6 20-bit range.</p>
Version info	Available as of DLL 600, OUT 600, RBF 600.
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	HeadPara = get_head_para(HeadNo, ParaNo)
Parameters	<p>HeadNo Number of the scan head connector. As an unsigned 32-bit value. Allowed values: = 1: First scan head connector. = 2: Second scan head connector.</p> <p>ParaNo Number of the parameter. As an unsigned 32-bit value. Allowed values: 0...15. Mapping: see Section "ct5 Correction File Header", page 177.</p>
Result	Parameter value, see Section "ct5 Correction File Header", page 177 . As a 64-bit IEEE floating point value.
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 "ct5 Correction File Header", page 177. If the parameters HeadNo and ParaNo are out of range, then the return value is 0 (get_last_error return code RTC6_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 (for example, for HeadNo = 2 if the Option "Second Scan Head Control" has not been enabled) and no 3D correction table has been assigned to the other head. If a 3D correction table has been assigned to a head, then this 3D correction table's parameter is returned regardless of HeadNo (two 3D correction tables cannot be simultaneously assigned). HeadNo must nevertheless be 1 or 2 (see preceding comment).
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	get_table_para



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>																																													
Parameters	<p>Head = 1: Returns the status of the first scan head connector (Byte #1 = Byte #0).</p> <p>= 2: Returns the status of the second scan head connector (Byte #1 = Byte #0).</p> <p>Else: Returns the status of the first scan head connector (Byte #0) and of the second scan head connector (Byte #1).</p>																																													
Result	<p>XY2-100 status word. As an unsigned 32-bit value.</p> <table> <tr> <td>Byte #0</td> <td>Bit #0</td> <td>1.</td> </tr> <tr> <td>(LSB)</td> <td>(LSB)</td> <td></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>Bit #8</td> <td>Bit assignments as with Byte #0.</td> </tr> <tr> <td></td> <td>...</td> <td></td> </tr> <tr> <td></td> <td>Bit #15</td> <td></td> </tr> <tr> <td>Byte #2</td> <td>Bit #16</td> <td>0.</td> </tr> <tr> <td>...</td> <td>...</td> <td></td> </tr> <tr> <td>Byte #3</td> <td>Bit #31</td> <td></td> </tr> </table>	Byte #0	Bit #0	1.	(LSB)	(LSB)			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	Bit #8	Bit assignments as with Byte #0.		...			Bit #15		Byte #2	Bit #16	0.		Byte #3	Bit #31	
Byte #0	Bit #0	1.																																												
(LSB)	(LSB)																																													
	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	Bit #8	Bit assignments as with Byte #0.																																												
	...																																													
	Bit #15																																													
Byte #2	Bit #16	0.																																												
...	...																																													
Byte #3	Bit #31																																													
Comments	<ul style="list-style-type: none"> • get_head_status is available even with the SL2-100 protocol, if the data type is not set to the 20-bit status word itself, see Section "Status Information Returned from the Scan System", page 182. • The status bits are returned by get_head_status in Bit #3...Bit #7 and/or Bit #11...Bit #15. Independently of the scan system's current state, Bit #0 and Bit #8 are returned by get_head_status as 1, while Bit #1 and Bit #9 are returned as 0. • If no scan system is currently connected or is not switched on, then the value 0 is returned. • The PowerOK signal is an electronically generated signal. It means: The scan system servo control is ready for input data ("PowerOK", actually corresponds to a "ServoOK"). Therefore, it cannot be used to check whether a connected scan head is switched on at all. get_startstop_info (Bit #17 and/or Bit #25) can be used for distinguishing. 																																													

Ctrl Command	get_head_status
Comments (cont'd)	<ul style="list-style-type: none"> 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. With <i>iDRIVE</i> scan systems (see Glossary entry on page 26) <ul style="list-style-type: none"> Without the SL2-100 interface, get_head_status only returns meaningful return values, if the XY2-100 status word has been 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 (for example, at Bit #3,4). 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 "Controlling 2D Scan Systems and 3D Scan Systems", page 235 for information about using two <i>scan heads</i>.
RTC4→RTC6	<p>Basically unchanged functionality. However:</p> <ul style="list-style-type: none"> The RTC6 allows the simultaneous reading of both scan head connectors' status words, and with <i>iDRIVE</i> scan systems with SL2-100 interface even independently of the signal to be returned (set by control_command). With <i>iDRIVE</i> scan systems, Bit #0 and Bit #1 do not return information about the operational readiness of the scan system (see get_value).
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	get_value , get_values , set_trigger , set_trigger4 , get_waveform

Ctrl Command	get_hex_version
Function	Returns the version number of the DSP program file RTC6OUT.out , which is currently loaded on the RTC6.
Call	<code>HexVersion = get_hex_version()</code>
Result	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 600...699. get_hex_version returns the following values: <ul style="list-style-type: none"> – if the Option "3D" is <i>not</i> enabled values in the range 2600...2699 (version number + 2000) – if the Option "3D" is <i>enabled</i> values in the range 3600...3699 (version number + 3000) The file name extension for RTC6 DSP program files is <code>*.out</code>. See also load_program_file. The software version number can also be returned after an RTC6_VERSION_MISMATCH or RTC6_ACCESS_DENIED error. The return value is 0 if no program has yet been loaded. The board-specific error variables LastError and AccError, see Chapter 6.8 "Error Handling", page 129, are neither generated nor altered by get_hex_version.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 first scan head connector.								
Call	<code>get_hi_data(&X1, &X2, &Y1, &Y2)</code>								
Returned parameter values	<table border="0"> <tr> <td>X1</td> <td>x1 coordinate of the currently stored (most recently measured) Home-In positions. In bits. As a pointer to a signed 32-bit value.</td> </tr> <tr> <td>X2</td> <td>Like X1 (analogously).</td> </tr> <tr> <td>Y1</td> <td>Like X1 (analogously).</td> </tr> <tr> <td>Y2</td> <td>Like X1 (analogously).</td> </tr> </table>	X1	x1 coordinate of the currently stored (most recently measured) Home-In positions. In bits. As a pointer to a signed 32-bit value.	X2	Like X1 (analogously).	Y1	Like X1 (analogously).	Y2	Like X1 (analogously).
X1	x1 coordinate of the currently stored (most recently measured) Home-In positions. In bits. As a pointer to a signed 32-bit value.								
X2	Like X1 (analogously).								
Y1	Like X1 (analogously).								
Y2	Like X1 (analogously).								
Comments	<ul style="list-style-type: none"> get_hi_data is synonymous with get_hi_pos with HeadNo = 1 (see comments there). 								
RTC4→RTC6	Unchanged functionality. In addition: increased value range. The returned values are in the 20-bit value range.								
RTC5→RTC6	Unchanged functionality.								
Version info	Available as of DLL 600, OUT 600, RBF 600.								
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>
Parameters	<p>HeadNo Number of the scan head connector. As an unsigned 32-bit value. Allowed values. = 1: First scan head connector. = 2: Second scan head connector.</p>
Returned parameter values	<p>X1 x1 coordinate of the currently stored (most recently measured) Home-In positions. In bits. As a pointer to a signed 32-bit value.</p> <p>X2 Like X1 (analogously).</p> <p>Y1 Like X1 (analogously).</p> <p>Y2 Like X1 (analogously).</p>
Comments	<ul style="list-style-type: none"> For information on using <code>get_hi_pos</code>, see Section "Customer-Specific Calibration", page 277. Make sure that the scan system currently attached to the specified scan head connector is the same scan system which has been 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: <ul style="list-style-type: none"> – no scan system equipped with automatic self-calibration (Home-In sensors) is attached to the specified scan head connector – an error has occurred during determination of the Home-In values for such a system Directly after initialization (init_rtc6_dll), particularly prior to a first call of auto_cal(<code>Command</code> = 0, 1 or 3), the returned values are the Home-In reference values stored in the Flash memory. If such reference values have not been successfully determined at least once by auto_cal(<code>Command</code> = 0), <code>get_hi_pos</code> returns 0. <code>get_hi_pos</code> is available even without (current) explicit access rights to a specific RTC6 board. However, the return values are 0 as long as no access to the addressed board has been successful at least once before. If parameter values are invalid, then all returned coordinates are 0 (get_last_error return code <code>RTC6_PARAM_ERROR</code>).
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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	<code>InputPointer = get_input_pointer()</code>
Result	Position of the input pointer [0...(2 ²³ -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 RTC6 list memory (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. <code>get_input_pointer</code> 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 <code>get_input_pointer</code> 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>LastError</code> and <code>AccError</code>, see Section "Error Handling", page 129, are neither generated nor altered by <code>get_input_pointer</code>.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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	<code>IOStatus = get_io_status()</code>
Result	16-bit value (DIGITAL OUT0...DIGITAL OUT15). As an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> <code>get_io_status</code> is designed for use in combination with set_io_cond_list and clear_io_cond_list. See also Section "Example Code (Pascal)", page 295. See also Section "16-Bit Digital Input Port and 16-Bit Digital Output Port", page 77.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	write_io_port , write_io_port_mask , set_io_cond_list , clear_io_cond_list



Ctrl Command	get_jump_table
Function	Reads out the Jump Delay table which is currently stored on the board. Then copies the 1024 corresponding unsigned 16-bit values to the specified PC address.
Call	<code>ErrorCode = get_jump_table(Addr)</code>
Parameters	Addr PC address for the 2048 byte memory area.
Result	<p>Error code. As an unsigned 32-bit value.</p> <p>0 No error. 11 RTC6 board driver error.</p>
Notes	<ul style="list-style-type: none"> Do not call get_jump_table 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→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
Reference	set_jump_table , load_jump_table_offset

Ctrl Command	get_lap_time
Function	Returns the <i>current RTC6 Timer</i> value (without resetting it to zero).
Call	<code>TimerValue = get_lap_time()</code>
Result	RTC6 Timer value in seconds since the last call of save_and_restart_timer . In seconds. As a 64-bit IEEE floating point value.
Comments	<ul style="list-style-type: none"> get_lap_time serves to query the elapsed time of a time consuming marking during processing.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	get_time , save_and_restart_timer

Ctrl Command	get_laser_pin_in
Function	Returns the current status of the 2-Bit-Digital digital input port at the LASER Connector , see also Section "2-Bit Digital Input Port", page 73 .
Call	<code>LaserPinIn = get_laser_pin_in()</code>
Result	As an unsigned 32-bit value. Bit #0 DIGITAL IN1. (LSB) Bit #1 DIGITAL IN2. Bit #2 Reserved. Bit 31 Reserved.
Comments	• –
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 get_error .
Comments	<ul style="list-style-type: none"> For error handling see Chapter 6.8 "Error Handling", page 129. get_last_error and n_get_last_error are available even without explicit access rights to a specific RTC6 board. The board-specific error variables LastError and AccError, see Chapter 6.8 "Error Handling", page 129, are neither generated nor altered by get_last_error. Each eth_get_last_error error also leads to a get_last_error error RTC6_ETH_ERROR.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	eth_get_last_error, get_error, reset_error, set_verify



Ctrl Command	get_list_pointer				
Function	Returns the current relative position of the input pointer as well as 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 absolute list memory address (offset relative to the start of "List 1") of the input pointer can be queried by get_input_pointer (see also comments there). The number of list positions until the end of the respective list (from the input pointer) can be queried by get_list_space. The board-specific error variables <code>LastError</code> and <code>AccError</code>, see Chapter 6.8 "Error Handling", page 129, are neither generated nor altered by get_list_pointer. If the input pointer is invalid when get_list_pointer is called (for example, after a list_return), the return values are <code>ListNo = 0</code> and <code>Pos = 0xFFFFFFF</code>. 				
RTC4→RTC6	New command.				
RTC5→RTC6	Unchanged functionality.				
Version info	Available as of DLL 600, OUT 600, RBF 600.				
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 a 64-bit IEEE floating point value.
Returned parameter values	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 get_list_serial is typically the one most recently marked by mark_serial or mark_serial_abs. For usage of get_list_serial, see Chapter 7.5.2 "Marking Serial Numbers", page 209.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 list memory area "List 3", then get_list_space returns the amount of still-available protected memory (otherwise the input pointer is not located in the protected area).
RTC4→RTC6	get_list_space has been made available on the RTC4 to support the RTC4-Circular Queue Mode and returns the distance between the input pointer and output pointer. The RTC6 does not support the RTC4-Circular Queue Mode . The input pointer position can be queried by get_input_pointer or get_list_pointer and the output pointer position can be queried by get_status or get_out_pointer .
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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. get_marking_info also returns the error bits of automatic suppression of laser control signals.
Call	MarkingInfo = get_marking_info()
Result	<p>Error code. As an unsigned 32-bit value.</p> <p>Bit #0 = 1: Processing-on-the-fly underflow in x direction ($X < -524,288$). (LSB)</p> <p>Bit #1 = 1: Processing-on-the-fly overflow in x direction ($X > +524,287$).</p> <p>Bit #2 = 1: Processing-on-the-fly underflow in y direction ($Y < -524,288$).</p> <p>Bit #3 = 1: Processing-on-the-fly overflow in y direction ($Y > +524,287$).</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: An error has occurred during activation of Processing-on-the-fly correction by activate_fly_2d, activate_fly_2d_encoder, activate_fly_xy or activate_fly_xy_encoder ("ActivateFlyError"). See also Chapter 8.6.7 "Synchronizing Processing-on-the-fly Applications", page 251, Chapter 8.6.8 "Encoder Resets", page 253 and Chapter 8.6.9 "Monitoring Processing-on-the-fly Corrections", page 254.</p> <p>Bit #10 PosAck error bit of scan system A, x axis.</p> <p>Bit #11 TempOK error bit of scan system A, x axis.</p> <p>Bit #12 PowerOK error bit of scan system A, x axis.</p> <p>Bit #13 PosAck error bit of scan system A, y axis.</p> <p>Bit #14 TempOK error bit of scan system A, y axis.</p> <p>Bit #15 PowerOK error bit of scan system A, y axis.</p> <p>Bit #16 = 1: Signal 1 of encoder input port ENCODER X has too-short spacing.</p> <p>Bit #17 = 1: Signal 2 of encoder input port ENCODER X has too-short spacing.</p> <p>Bit #18 = 1: Signal 1 of encoder input port ENCODER Y has too-short spacing.</p> <p>Bit #19 = 1: Signal 2 of encoder input port ENCODER Y has too-short spacing.</p> <p>Bit #20 = 1: Improper signal sequence at encoder input port ENCODER X.</p> <p>Bit #21 = 1: Improper signal sequence at encoder input port ENCODER Y.</p> <p>Bit #22 = 1: Processing-on-the-fly underflow in z direction ($Z < -524,288$).</p> <p>Bit #23 = 1: Processing-on-the-fly overflow in z direction ($Z > +524,287$).</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 scan system B, x axis.</p> <p>Bit #27 TempOK error bit of scan system B, x axis.</p> <p>Bit #28 PowerOK error bit of scan system B, x axis.</p> <p>Bit #29 PosAck error bit of scan system B, y axis.</p> <p>Bit #30 TempOK error bit of scan system B, y axis.</p> <p>Bit #31 PowerOK error bit of scan system B, y axis.</p>
Comments	<ul style="list-style-type: none"> For usage of <code>get_marking_info</code> and of the error bits Bit #0...Bit #7, see Chapter 8.6.9 "Monitoring Processing-on-the-fly Corrections", page 254. The limits for the customer-defined monitoring range are determined for: <ul style="list-style-type: none"> Xmin, Xmax, Ymin, Ymax (Bit #4...Bit #7) by <code>set_fly_limits</code> Zmin, Zmax (Bit #24...Bit #25) by <code>set_fly_limits_z</code> The error bits Bit #4...Bit #7 and Bit #24...Bit #25: <ul style="list-style-type: none"> Are <i>not</i> reset by <code>get_marking_info</code> Get implicitly reset by the conditional commands Can also be explicitly reset by <code>clear_fly_overflow</code> and <code>clear_fly_overflow_ctrl</code> 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. The error bits Bit #10...Bit #15 and Bit #26...Bit #31 are only set in case of an error if automatic suppression of laser control signals has been activated, see Section "Automatic Suppression of Laser Control Signals", page 186. 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 automatic suppression of laser control signals by <code>set_laser_control</code>. All error bits are reset by <code>get_marking_info</code>. All error bits are reset during initialization (by <code>load_program_file</code>). All error bits (except Bit #4...Bit #7 and Bit #24...Bit #25): <ul style="list-style-type: none"> Are reset when reading out by <code>get_marking_info</code> However, can be reset at any time as long as the error condition still persists
RTC4→RTC6	Unchanged functionality for info bits that are also used on the RTC4.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	<code>set_fly_x</code> , <code>set_fly_y</code> , <code>set_fly_z</code> , <code>set_fly_rot</code> , <code>set_fly_x_pos</code> , <code>set_fly_y_pos</code> , <code>set_fly_rot_pos</code> , <code>set_fly_limits</code> , <code>set_fly_limits_z</code> , <code>clear_fly_overflow</code> , <code>clear_fly_overflow_ctrl</code> , <code>if_not_activated</code>



Ctrl Command	get_master_slave
Function	Returns the master/slave status of the addressed RTC6 board.
Call	MasterSlaveStatus = get_master_slave()
Result	<p>Master/slave status. As an unsigned 32-bit value. Information whether a further RTC6 board is connected to the Master or Slave connector of the addressed RTC6 board:</p> <p>Bit #0 = 1: A RTC6 board is connected to the Slave connector. (LSB)</p> <p>Bit #1 = 1: A RTC6 board is connected to the Master connector.</p> <p>Bit #2 = 0.</p> <p>... ...</p> <p>Bit #31 = 0.</p> <p>Information, whether the addressed board is operated as a master, slave or single board:</p> <p>= 0 Single board.</p> <p>= 1 Slave without any further downstream slave board.</p> <p>= 2 Master.</p> <p>= 3 Slave together with a downstream slave board.</p>
Comments	<ul style="list-style-type: none"> See Chapter 6.6.3 "Master/Slave Operation", page 123. get_master_slave only returns the status, if the addressed RTC6 board has been previously initialized by load_program_file. Otherwise, 0 is returned.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	sync_slaves

Ctrl Command	get_mcbsp
Function	Returns the most recent input value that has been fully transferred by the McBSP 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→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	read_mcbsp

Undelayed Short List Command	get_mcbsp_list
Function	No function.
Call	<code>get_mcbsp_list()</code>
Comments	<ul style="list-style-type: none"> • get_mcbsp_list has no effect on the user program.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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"> • get_out_pointer 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→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	get_status , get_input_pointer , get_list_pointer



Ctrl Command	get_overrun
Function	Returns the number of overruns of the 10 μ s clock cycle since the last call and resets the overrun counter.
Call	NumberOfOverruns = get_overrun()
Result	Number of overruns of the 10 μ s clock cycle 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 181.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	–



Ctrl Command	get_RTC_mode
Function	Returns the currently set operation mode of the RTC6 DLL .
Call	<code>DLLMode = get_RTC_mode()</code>
Result	<p>DLL operation mode as a 32-bit value.</p> <p>= 4: RTC4 Compatibility Mode.</p> <p>= 5: RTC5 Compatibility Mode.</p> <p>= 6: RTC6 Standard Mode (default setting).</p>
Comments	<ul style="list-style-type: none"> The RTC6 DLL operation mode can be set by set_RTC4_mode, set_RTC5_mode or set_RTC6_mode. The default setting is RTC6 Standard Mode. get_RTC_mode is available even without explicit access rights to a particular RTC6 board. get_RTC_mode is not available as a multi-board command. The board-specific error variables LastError and AccError, see Chapter 6.8 "Error Handling", page 129, are neither generated nor altered by get_RTC_mode.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_RTC4_mode , set_RTC5_mode , set_RTC6_mode

Ctrl Command	get_RTC_version
Function	Returns the version number of the FPGA firmware (RTC6RBF.rbf) and information about enabled options of the RTC6 board.
Call	<code>RTCVersion = get_RTC_version()</code>
Result	<p>Version number of the FPGA firmware and additional information. As an unsigned 32-bit value.</p> <p>Bit #0 (LSB) Version number of the FPGA firmware (RTC6RBF.rbf). ... Bit #7 Bit #8 = 1: Option Processing-on-the-fly is enabled. See also Chapter 8.6 "Processing-on-the-fly", page 241. Bit #9 = 1: Option "Second Scan Head Control" is enabled. See also Section "2. SCANHEAD Socket Connector", page 67. Bit #10 = 1: Option "3D" is enabled. See also Chapter 8.5.2 "3D Scan Systems", page 236. Bit #11 = 1: Option "LDSA" is enabled (as of DLL 628). Bit #12 = 1: Option "SCANa" is enabled (as of DLL 605). Bit #13 = 1: Option "UFPM" is enabled (as of DLL 605). Bit #14 = 1: Option "syncA" is enabled (as of DLL 607). Bit #15 Reserved. Bit #16 DSP version number. ... Bit #23 Bit #24 Subversion number of the FPGA firmware (RTC6RBF.rbf). ... Bit #31</p>
Comments	<ul style="list-style-type: none"> The FPGA firmware version numbers are in the range 600...699. get_RTC_version(Bit #0..Bit #7) returns values in the range 0...99 (version number – 600). The current DSP version number is 3. The current FPGA firmware subversion is 0. The FPGA firmware version can even be returned after an RTC6_VERSION_MISMATCH or RTC6_ACCESS_DENIED error. The return value is 0, if no program has yet been loaded.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	get_hex_version , get_dll_version

Ctrl Command	get_scanahead_params
Comments	<ul style="list-style-type: none"> Only for scan systems with SCANAhead technology, for example, the excelliSCAN. This command is described in the "excelliSCAN Scan Heads – Functional Principle of SCANAhead Servo Control and Operation by RTC6 Boards" Manual.

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	CurrentSerialNo = <code>get_serial()</code>
Result	Serial number. As a 64-bit IEEE floating point value.
Comments	<ul style="list-style-type: none"> For usage of <code>get_serial</code>, see Chapter 7.5.2 "Marking Serial Numbers", page 209. <code>get_serial</code> should not be confused with <code>get_serial_number</code>, which returns the product serial number of the RTC6 board.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	select_serial_set

Ctrl Command	get_serial_number
Function	Returns the individual serial number of the active RTC6 board.
Call	<code>RTCSerialNumber = get_serial_number()</code>
Result	RTC6 serial number. As an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> Serial numbers of installed boards are ascertained by init_rtc6_dll and cached in the RTC6 DLL, from where you can query them by <code>get_serial_number</code>. <code>get_serial_number</code> is helpful when using several RTC6 boards in one computer (see Chapter 6.6 "Using Several RTC6 PCIe Boards in One PC", page 122). The associated multi-board command <code>n_get_serial_number</code> can be used for determining the relationship between the installed boards and the RTC6 DLL-internal numbers assigned to them during initialization. The RTC6 DLL-internal numbers are newly assigned during each initialization of a user program (see init_rtc6_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 rtc6_count_cards. <code>get_serial_number</code> and <code>n_get_serial_number</code> is available even without (current) explicit access rights to a specific RTC6 board. However, the return values are 0 as long as no access to the addressed board has been successful at least once before. The board-specific error variables <code>LastError</code> and <code>AccError</code> (see Chapter 6.8 "Error Handling", page 129) are neither generated nor altered by <code>get_serial_number</code>.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	rtc6_count_cards



Ctrl Command	get_standby
Function	Returns the currently set standby parameters.
Call	<code>get_standby(&HalfPeriod, &PulseLength)</code>
Returned parameter values	<p>HalfPeriod <i>Half of the currently set standby output period of the standby pulses.</i> As a pointer to an unsigned 32-bit value. 1 bit equals 1/64 μs.</p> <p>PulseLength <i>Currently set pulse length of the standby pulses.</i> As a pointer to an unsigned 32-bit value. 1 bit equals 1/64 μs.</p>
Comments	<ul style="list-style-type: none"> For usage of get_standby, see Section "Signals for "Laser Standby" Operation", page 185.
RTC4→RTC6	<p>New command.</p> <p>In RTC4 Compatibility Mode, the RTC6 divides the values for <code>HalfPeriod</code> and <code>PulseLength</code> by 8.</p>
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_standby

Ctrl Command	get_startstop_info
Function	Provides information about internal and External Starts and External Stops since the last call of get_startstop_info . Also provided are the current External Start and External 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 internal start has been executed (by execute_list or similar) since the last call of get_startstop_info.</p> <p>Bit #1 = 1: An External Start has been executed (by /START, /START2, /Slave-START, simulate_ext_start or simulate_ext_start_ctrl) since the last call of get_startstop_info.</p> <p>Bit #2 = 1: An internal stop has been executed (by stop_execution) since the last call of get_startstop_info.</p> <p>Bit #3 = 1: An External Stop has been executed (by /STOP, /STOP2, /Slave-STOP or simulate_ext_stop) since the last call of get_startstop_info.</p> <p>Bit #4 Ext-stop status (= logical AND operation of the signals /STOP, /STOP2, /Slave-STOP and simulate_ext_stop, see Figure 68): = 1: <i>No</i> stop signals are currently present at the input ports or the input ports are not connected. = 0: There is a stop signal at least at one of the input ports.</p> <p>Bit #5 \geq DLL 619: = 1: The timeout with wait_for_timestamp_counter_mode and wait_for_timestamp_counter_long.</p> <p>Bit #6 Reserved.</p> <p>Bit #7 Reserved.</p> <p>Bit #8 Reserved.</p> <p>Bit #9 = 1: The laser control signals are globally enabled, see Chapter 7.4.1 "Enabling, Activating and Switching Laser Control Signals", page 183. See also set_laser_control.</p> <p>Bit #10 = 1: The TTL laser control signals at the LASER1 and LASER2 output ports are active-LOW (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 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 status (= logical AND operation of the signals /START, /START2 and /Slave-START, see Figure 68): = 1: <i>No</i> start signals are currently present at the input ports or the input ports are not connected. = 0: A start signal is present at least at one of the input ports.</p> <p>Bit #13 = 1: The TTL laser control signal at the LASERON output port is active-LOW (the signal level can be defined by set_laser_control).</p>

Ctrl Command	get_startstop_info
Result (cont'd)	<p>Bit #14 = 1: The laser control signals are enabled (enable_laser). = 0: The laser control signals are disabled (disable_laser).</p> <p>Bit #15 = 1: If previously activated by eth_configure_link_loss: An Ethernet Link Loss has been detected since the last call of get_startstop_info.</p> <p>Bit #16 ... Bit #31 The error bits. Get set when an error occurs during data transmission from the scan system:</p> <ul style="list-style-type: none"> Bit #16...Bit #23 for the first scan head connector, Bit #24...Bit #31 for the second scan head connector. <p>Bit #16, Bit #24: Incorrect number of frames within a data block.</p> <p>Bit #17, Bit #25: Incorrect pulse length of signal received from scan system, maybe no scan system is connected.</p> <p>Bit #18, Bit #26: Preamble sequence incorrect.</p> <p>Bit #19, Bit #27: Bit count within a subframe incorrect.</p> <p>Bit #20, Bit #28: Parity error when reading data received from scan system.</p> <p>Bit #21, Bit #29: The present data is invalid (old).</p> <p>Bit #22, Bit #30: Reserved.</p> <p>Bit #23, Bit #31: Reserved.</p>
Comments	<ul style="list-style-type: none"> After get_startstop_info has been executed, reset are: <ul style="list-style-type: none"> The info bits Bit #0...Bit #3, Bit #15 The error bits Bit #5, Bit #16...Bit #31 See also Section "External Stop", page 288 and Section "External Start", page 289.
RTC4→RTC6	Unchanged functionality for info bits that are also used on the RTC4.
RTC5→RTC6	Unchanged functionality for info bits that are also used on the RTC5.
Version info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 622, OUT 622: Bit #15.
References	get_counts , get_status , set_control_mode

Ctrl Command	get_status																																	
Function	Returns the current List Execution Status values and the current (or most recent) position of the output pointer.																																	
Call	get_status(&Status, &Pos)																																	
Returned parameter values	<p>Status</p> <p>Status value. As a pointer to an unsigned 32-bit value.</p> <table> <tr> <td>Bit #0</td> <td>= 1: BUSY list execution status set. (LSB)</td> </tr> <tr> <td>Bit #1</td> <td>Reserved.</td> </tr> <tr> <td>...</td> <td></td> </tr> <tr> <td>Bit #6</td> <td></td> </tr> <tr> <td>Bit #7</td> <td>= 1: INTERNAL-BUSY list execution status set.</td> </tr> <tr> <td>Bit #8</td> <td>Reserved.</td> </tr> <tr> <td>...</td> <td></td> </tr> <tr> <td>Bit #14</td> <td></td> </tr> <tr> <td>Bit #15</td> <td>= 1: PAUSED list execution status set.</td> </tr> <tr> <td>Bit #16</td> <td>Reserved.</td> </tr> <tr> <td>...</td> <td></td> </tr> <tr> <td>Bit #22</td> <td></td> </tr> <tr> <td>Bit #23</td> <td>= 1: HEAD_BUSY status set. Operating status for excelliSCAN, see "excelliSCAN Scan Heads – Functional Principle of SCANAhead Servo Control and Operation by RTC6 Boards" Manual.</td> </tr> <tr> <td>Bit #24</td> <td>Reserved.</td> </tr> <tr> <td>...</td> <td></td> </tr> <tr> <td>Bit #31</td> <td></td> </tr> </table>	Bit #0	= 1: BUSY list execution status set. (LSB)	Bit #1	Reserved.	...		Bit #6		Bit #7	= 1: INTERNAL-BUSY list execution status set.	Bit #8	Reserved.	...		Bit #14		Bit #15	= 1: PAUSED list execution status set.	Bit #16	Reserved.	...		Bit #22		Bit #23	= 1: HEAD_BUSY status set. Operating status for excelliSCAN, see " excelliSCAN Scan Heads – Functional Principle of SCANAhead Servo Control and Operation by RTC6 Boards " Manual .	Bit #24	Reserved.	...		Bit #31		<p>Pos</p> <p>Current (or most recent) position of the output pointer (absolute memory address). As a pointer to an unsigned 32-bit value.</p>
Bit #0	= 1: BUSY list execution status set. (LSB)																																	
Bit #1	Reserved.																																	
...																																		
Bit #6																																		
Bit #7	= 1: INTERNAL-BUSY list execution status set.																																	
Bit #8	Reserved.																																	
...																																		
Bit #14																																		
Bit #15	= 1: PAUSED list execution status set.																																	
Bit #16	Reserved.																																	
...																																		
Bit #22																																		
Bit #23	= 1: HEAD_BUSY status set. Operating status for excelliSCAN, see " excelliSCAN Scan Heads – Functional Principle of SCANAhead Servo Control and Operation by RTC6 Boards " Manual .																																	
Bit #24	Reserved.																																	
...																																		
Bit #31																																		
Comments	<ul style="list-style-type: none"> For a description of when the BUSY list execution status, INTERNAL-BUSY list execution status or PAUSED list execution status values are set or not set, see Chapter 6.4.3 "List Execution Status", page 107. (BUSY list execution status and PAUSED list execution status set) requires restart_list for continuation, (BUSY list execution status not set and PAUSED list execution status set) requires release_wait and (both BUSY list execution status and PAUSED list execution status not set) requires execute_list_pos. "Continuation" is not allowed with (BUSY list execution status set and PAUSED list execution status not set) and a currently running list. An improper continuation generates the get_last_error return code RTC6_BUSY. With (INTERNAL-BUSY list execution status set), release_wait and execute_list_pos are only executed with a delay (after INTERNAL-BUSY list execution status has been reset again). 																																	

Ctrl Command	get_status
Comments (cont'd)	<ul style="list-style-type: none"> 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 list memory area "List 3", the output pointer's position <code>Pos</code> is queried, then the position is returned of the list command in the list memory area ("List 1" or "List 2") in which the output pointer most recently resided (typically from where the subroutine has been called (for example, with <code>list_call</code>)). <code>pause_list</code> and <code>set_wait</code> leave <code>Pos</code> unchanged. <code>get_status</code> 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 <code>get_out_pointer</code>. The current input pointer position can be queried by <code>get_input_pointer</code> or <code>get_list_pointer</code>. List Status values for individual lists can be queried by <code>read_status</code>. <code>get_status</code>, <code>get_input_pointer</code> and <code>read_status</code> can be used during loading of a list to ensure that no list is overwritten that has still not been processed (see also the <code>load_list</code>). As long as no program has been loaded (by <code>load_program_file</code>), <code>get_status</code> returns undefined values. <code>get_head_status</code> is available for querying the status signals of the scan heads.
RTC4→RTC6	<p>Basically unchanged functionality. However:</p> <p>The parameter <code>Status</code> returns additionally the INTERNAL-BUSY list execution status and PAUSED list execution status with an RTC6.</p>
RTC5→RTC6	<p>Changed functionality.</p> <ul style="list-style-type: none"> The <code>HEAD_BUSY</code> status is returned by Bit #23.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	<code>read_status</code> , <code>get_input_pointer</code> , <code>get_list_pointer</code> , <code>get_out_pointer</code>

Ctrl Command	get stepper status																														
Function	Returns the following status information for both stepper motor output ports: the current statuses of the stepper motor signals, the currently defined CLOCK pulse period, status "Busy" and status "Init", and the current values of the internal position variables.																														
Call	<code>get stepper_status(&Status1, &Pos1, &Status2, &Pos2)</code>																														
Returned parameter values	<table> <tr> <td>Status1</td> <td>Current status of stepper motor output 1. As a pointer to an unsigned 32-bit value.</td> </tr> <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 = limit switch signal.</td> </tr> <tr> <td> Bit #4</td> <td>Status "Busy".</td> </tr> <tr> <td> Bit #5</td> <td>Status "Init".</td> </tr> <tr> <td> Bit #6</td> <td>Reserved.</td> </tr> <tr> <td> Bit #7</td> <td>Reserved.</td> </tr> <tr> <td> Bit #8</td> <td>CLOCK pulse period (24-bit value).</td> </tr> <tr> <td> ...</td> <td></td> </tr> <tr> <td> Bit #31</td> <td></td> </tr> </table> <table> <tr> <td>Pos1</td> <td>Current value of the internal position variable for stepper motor output port 1. As a pointer to a signed 32-bit value.</td> </tr> <tr> <td>Status2</td> <td>Current status of stepper motor output port 1. Otherwise, like Status1.</td> </tr> <tr> <td>Pos2</td> <td>Like Pos1.</td> </tr> </table>	Status1	Current status of stepper motor output 1. As a pointer to an unsigned 32-bit value.	Bit #0	ENABLE signal. (LSB)	Bit #1	DIRECTION signal.	Bit #2	CLOCK signal.	Bit #3	SWITCH signal = limit switch signal.	Bit #4	Status "Busy".	Bit #5	Status "Init".	Bit #6	Reserved.	Bit #7	Reserved.	Bit #8	CLOCK pulse period (24-bit value).	...		Bit #31		Pos1	Current value of the internal position variable for stepper motor output port 1. As a pointer to a signed 32-bit value.	Status2	Current status of stepper motor output port 1. Otherwise, like Status1.	Pos2	Like Pos1.
Status1	Current status of stepper motor output 1. As a pointer to an unsigned 32-bit value.																														
Bit #0	ENABLE signal. (LSB)																														
Bit #1	DIRECTION signal.																														
Bit #2	CLOCK signal.																														
Bit #3	SWITCH signal = limit switch signal.																														
Bit #4	Status "Busy".																														
Bit #5	Status "Init".																														
Bit #6	Reserved.																														
Bit #7	Reserved.																														
Bit #8	CLOCK pulse period (24-bit value).																														
...																															
Bit #31																															
Pos1	Current value of the internal position variable for stepper motor output port 1. As a pointer to a signed 32-bit value.																														
Status2	Current status of stepper motor output port 1. Otherwise, like Status1.																														
Pos2	Like Pos1.																														
Comments	<ul style="list-style-type: none"> For programming the stepper motor signals, see Chapter 9.1.5 "Controlling Stepper Motors", page 283. If the SWITCH signal (limit switch signal) is set to (pin is LOW), no more CLOCK pulses are generated. Status "Busy" indicates that a previously initiated (by <code>stepper_abs</code>, <code>stepper_rel</code>, etc.) set-position movement has not yet completed. Status "Init" indicates that a previously initiated (by <code>stepper_init</code>) reference movement has not yet completed. 																														
RTC4→RTC6	New command.																														
RTC5→RTC6	Unchanged functionality.																														
Version info	Available as of DLL 600, OUT 600, RBF 600.																														
References	–																														



Ctrl Command	get_sub_pointer
Function	Returns the absolute start address of an indexed subroutine.
Call	SubPointer = <code>get_sub_pointer(Index)</code>
Parameters	Index Index of the indexed subroutine. As an unsigned 32-bit value. Allowed value range: [0...1023].
Result	Absolute start address. As an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> The <code>get_sub_pointer</code> 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 list memory area "List 3" depends on whether the subroutine has been loaded into the protected list memory area "List 3" or an unprotected subroutine has been only subsequently referenced. If <code>Index > 1023</code> or if no subroutine has been referenced with the specified index, then <code>get_sub_pointer</code> returns the value "-1" (for example, $2^{32}-1$). 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 <code>save_disk/load_disk</code> might alter the absolute memory address.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	get_char_pointer , get_text_table_pointer

Ctrl Command	get_sync_status
Function	Returns the master/slave synchronization information on the addressed RTC6 board.
Call	MasterSlaveSyncStatus = get_sync_status()
Result	<p>Master/slave synchronization information. As an unsigned 32-bit value.</p> <p>Bit #0 Master/slave synchronization status [0...640]. As unsigned 10 bit value. ... Bit #9 < 4: The addressed board is synchronous to the master board (or to its preceding board in the master/slave chain). 640: The addressed board is operated as master.</p> <p>Bit #10 = 0: Bit #0...Bit #9 contain valid values. = 1: Bit #0...Bit #9 do not contain valid values. No External Start has been received from the master board yet.</p> <p>Bit #11 Reserved.</p> <p>Bit #12 = 1: A master board has been detected. However, no /Slave-START or /Slave-STOP can be received. See master_slave_config.</p> <p>Bit #13 = 1: A Slave board has been detected. However, no /Slave-START or /Slave-STOP can be received. See master_slave_config.</p> <p>Bit #14 = 1: Since the last call of get_sync_status, the connection to the master has been lost.</p> <p>Bit #15 = 1: Since the last call of get_sync_status, the connection to the slave has been lost.</p> <p>Bit #16 Reserved. ... Bit #20</p> <p>Bit #21 Exact propagation time in 1/64 μs clock cycles between two RTC6 boards (outbound and return). As unsigned 10 bit value. ... Bit #30</p> <p>Bit #31 = 1: The propagation time Bit #21...Bit #30) could not be measured successfully.</p>

Ctrl Command	get_sync_status
Comments	<ul style="list-style-type: none"> For usage of <code>get_sync_status</code>, see Chapter 6.6.3 "Master/Slave Operation", page 123. If the addressed board has not been initialized previously by <code>load_program_file</code>, the <code>get_sync_status</code> returns 0. As of RBF 619: <ul style="list-style-type: none"> Synchronization is automatic. <code>sync_slaves</code> is no longer required. The master/slave synchronization state is measured automatically. The exact propagation time is returned by <code>Bit #21...Bit #30</code>, if <code>Bit #31</code> does not indicate an error. Up to RBF 618: <ul style="list-style-type: none"> Prior to <code>get_sync_status</code>, the boards of the master/slave chain should have been synchronized by <code>sync_slaves</code>. To ensure that <code>get_sync_status</code> actually returns the current master/slave synchronization status, you should first have already triggered an External Start for the master board by an external start signal or by <code>simulate_ext_start_ctrl</code>, see Section "External Start", page 289. For all slave boards for which External Starts and External Stops are not suppressed (see <code>master_slave_config</code>), this start then triggers a measurement of the time difference between the respective /Slave-START pulse and respective 10 μs clock cycle. This time difference gets stored on each board as the master/slave synchronization status (in units of 1/64 μs) and remains stored there until the a new External Start is triggered for the master board. This gives you the flexibility to query the synchronization status by <code>get_sync_status</code> 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 Chapter 6.6.3 "Master/Slave Operation", page 123: master/slave synchronization status < 3. In a not-explicitly-synchronized state (prior to <code>sync_slaves</code>), the master/slave synchronization status might coincidentally be < 3. If so, then the boards behave as if synchronized. If <code>Bit #12</code> or <code>Bit #13</code> is set, the affected cards cannot be synchronized with <code>sync_slaves</code>. If <code>Bit #14</code> or <code>Bit #15</code> is set, /Slave STARTs or /Slave STOPs may not have been forwarded. In this case it is recommended to adjust <code>master_slave_config</code> and to execute <code>sync_slaves</code> again. If one or more of <code>Bit #12...Bit #15</code> are sporadically 1, this may indicate electromagnetic disturbances that interfere with communication between the boards.
RTC4→RTC6	New command.
RTC5→RTC6	Basically unchanged functionality. However, more details are available.
Version info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 614, OUT 614, RBF 619: internal improvements and additional results.
References	<code>sync_slaves</code> , <code>get_master_slave</code> , <code>master_slave_config</code>



Ctrl Command	get_table_para
Function	Returns the value of the specified parameter from a currently loaded correction table.
Call	TablePara = get_table_para(TableNo, ParaNo)
Parameters	<p>TableNo Number of the currently loaded correction table. As an unsigned 32-bit value. Allowed values: [1...8]. See also number_of_correction_tables.</p> <p>ParaNo Number of the parameter. As an unsigned 32-bit value. Allowed values: 0...15. Assignment see Section "ct5 Correction File Header", page 177.</p>
Result	Parameter value, see Section "ct5 Correction File Header", page 177 . As a 64-bit IEEE floating point value.
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 "ct5 Correction File Header", page 177. If the parameters TableNo and ParaNo are out of range, then the return value is 0 (get_last_error return code RTC6_PARAM_ERROR). If no correction table with the specified number has been loaded, then the parameter values are undefined.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	–

Ctrl Command	get_temperature
Function	Returns the current temperature of the RTC6 board.
Call	Temp = get_temperature()
Parameters	None.
Result	Temperature of the RTC6 board. In degrees Celsius. As a 64-bit IEEE floating point value.
Comments	<ul style="list-style-type: none"> Execution of get_temperature takes about 1 ms. Continuous readout of the analog voltage (after a read_analog_in call) is delayed by get_temperature by about 0.1 ms, see Section "Analog Input Ports", page 85.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 631, OUT 632.
References	–

Ctrl Command	get_text_table_pointer
Function	Returns the absolute start address of an indexed text string.
Call	TextTablePointer = get_text_table_pointer(Index)
Parameters	Index Index of the indexed text string. As an unsigned 32-bit value. Allowed value 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 protected list memory area "List 3" depends on whether the text string has been loaded into the protected list memory area "List 3" or an unprotected subroutine has been only subsequently referenced. If Index > 41 or if no text string has been referenced with the specified index, then get_text_table_pointer returns the value "-1" (for example, $2^{32}-1$). 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, for example, 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 list_call_cond is ignored.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	get_char_pointer, get_sub_pointer

Ctrl Command	get_time
Function	Returns the RTC6 Timer value (without resetting it to zero). It has been stored during the most recent call of save_and_restart_timer .
Call	TimerValue = get_time()
Result	RTC6 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→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	get_lap_time, save_and_restart_timer



Ctrl Command	get_timestamp_long
Function	Returns the current value of the 64-bit "Timestamp Counter".
Call	get_timestamp_long(&TimeStampL, &TimeStampH)
Returned parameter values	TimeStampL Lower 64-bit "Timestamp Counter" value. As an unsigned 32-bit value.
	TimeStampH Upper 64-bit "Timestamp Counter" value. As an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> The RTC6 board-internal 64-bit "Timestamp Counter" TimeStampCounterLong is: <ul style="list-style-type: none"> Initialized with 0 at load_program_file and then keeps running until the next load_program_file Used with wait_for_timestamp_counter_long The lower part of TimeStampCounterLong (= returned parameter value TimeStampL) and the 32-bit "Timestamp Counter" are identical. See Chapter 8.12 "Time Measurements", page 280.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 624, OUT 624.
References	wait_for_timestamp_counter_long



Ctrl Command	get_transform																																	
Function	Transfers to the PC the position values that were recorded by set_trigger and stored on the RTC6, and applies backward transformation to these values.																																	
Call	get_transform(Number, Ptr1, Ptr2, Ptr, Code)																																	
Parameters	<p>Number Number [1...2²³] 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, an unsigned 32-bit value or unsigned 64-bit value) to the two areas of PC main memory to which the backward transformed values should be transferred (see also Code).</p> <p>Ptr2</p> <p>Ptr Pointer (in C and C++ data type ULONG_PTR, an unsigned 32-bit value or unsigned 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 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). As an unsigned 32-bit value.</p> <p>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.</td> </tr> <tr> <td></td> <td></td> <td>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.</td> </tr> <tr> <td></td> <td></td> <td>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>Bit #6</td> <td></td> <td>Reserved.</td> </tr> <tr> <td></td> <td>...</td> <td></td> </tr> <tr> <td></td> <td>Bit #31</td> <td></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.	Bit #6		Reserved.		...			Bit #31	
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.																																
Bit #6		Reserved.																																
	...																																	
	Bit #31																																	

Ctrl Command	get_transform	
Parameters (cont'd)	Code (cont'd)	<p>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>Bit #1 = 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>Bit #5 Reserved.</p> <p>...</p> <p>Bit #31</p>
Comments	<ul style="list-style-type: none"> For backward transformation of position values, see Chapter 8.1.3 "Monitoring the Positioning", page 213. <code>get_transform(Number, Ptr1, Ptr2, Ptr, Code)</code> transfers to the PC the data pairs recorded by set_trigger by executing <code>get_waveform(1,Number,Ptr1)</code> and <code>get_waveform(2,Number,Ptr2)</code> and overwrites the data pairwise by using <code>transform(Sig1,Sig2,Ptr,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 has been 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 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 Bit #1 is appropriately set. 	



Ctrl Command	get_transform
Comments (cont'd)	<ul style="list-style-type: none"> If the <code>get_transform</code> 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>RTC6_PARAM_ERROR</code>). The same applies for <code>Number = 0</code> or <code>Number > 2²³</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> (for example, 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>RTC6_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 has been 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>RTC6_SEND_ERROR</code>.
RTC4→RTC6	<p>New command.</p> <p>Even in the RTC4 Compatibility Mode, all coordinate values transferred to the PC are in the RTC6 20-bit range. The backward transformed coordinate values must, if necessary, be divided by 16 by users themselves.</p>
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	upload_transform , transform , set_trigger , get_waveform

Ctrl Command	get_value
Function	Returns the current value of the specified signal.
Call	<code>Value = get_value(Signal)</code>
Parameters	<p>Signal Desired signal type. As an unsigned 32-bit value.</p>
Result	<p>Current value of the specified signal. As a signed 32-bit value.</p>
Comments	<ul style="list-style-type: none"> The selectable signal types are identical to those of <code>set_trigger</code> (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 <code>get_value</code> does not read out a signal and returns 0 (<code>get_last_error</code> return code <code>RTC6_PARAM_ERROR</code>). To observe the specified signal over a long time period, use <code>set_trigger</code> to start a corresponding measurement session. When using an <i>iDRIVE</i> scan system (see Glossary entry on page 26), 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. For <code>Signal = 0</code>, <code>get_value</code> returns the current laser status (LASERON signal) even when list execution has already been finished. When you query data returned as status signals from the scan system to the RTC6 (<code>Status<AX...BY></code>), then be mindful of the returned data type's value range when evaluating it (see <code>control_command</code>): <ul style="list-style-type: none"> – Data types originally generated in the scan system as unsigned 16-bit values (for example, the <code>XY2-100 status word</code> or the serial number) and returned to the RTC6 as unsigned 20-bit values (whereby Bit #0...Bit #3 = 0) contain the relevant information in Bit #4...Bit #19. Here, only Bit #4...Bit #19 should be evaluated (see code example below). For Bit #20...Bit #31 of this data type, <code>get_value</code> returns to the PC not only zero, but (depending on Bit #19 of the underlying 16-bit status value) even the value one. So only evaluate Bit #4...Bit #19. – In contrast, data types returned to the RTC6 as signed 20-bit values (for example, actual positions or actual speeds) can be evaluated as a complete signed 32-bit value returned by <code>get_value</code> (see also code example below).



Ctrl Command	get_value
Example (C/C++)	<p>Querying diverse data types (first 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) Only with iDRIVE systems: Serial number</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) Only with iDRIVE systems: Actual position</p> <pre>long real_position; control_command (1, 1, 0x0501); real_position = get_value(1);</pre>
RTC4→RTC6	<p>Basically unchanged functionality. However:</p> <p>Even in RTC4 Compatibility Mode, all returned values are in the RTC6 20-bit range, but are transferred to the PC as 32-bit values (see above).</p>
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	get_values , set_trigger , get_waveform , get_head_status



Ctrl Command	get_values
Function	Returns the current values of up to 4 specified signals.
Call	<code>get_values(SignalPtr, ResultPtr)</code>
Parameters	<p>SignalPtr Pointer (in C and C++ data type <code>ULONG_PTR</code>, an unsigned 32-bit value or unsigned 64-bit value) to an array of 4 unsigned 32-bit values, where the to-be-outputted signal types are specified.</p> <p>ResultPtr Pointer (in C and C++ data type <code>ULONG_PTR</code>, an unsigned 32-bit value or unsigned 64-bit value) to an array of 4 signed 32-bit values, where the current values of the up to 4 specified signals are to be stored.</p>
Comments	<ul style="list-style-type: none"> Up to 4 desired signals can be simultaneously queried. The selectable signal types are identical to those of <code>set_trigger</code> (refer to the comments there for the allowed value range, signal types and other information). The to-be-outputted 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 <code>get_value</code> (see comments there). <code>get_values</code> returns 0 and performs no query on channels for which an invalid signal type has been specified by <code>SignalPtr</code>. A <code>get_last_error</code> return code <code>RTC6_PARAM_ERROR</code> is only generated if all 4 specified signal types are invalid. If any of the pointer parameters are <code>NULL</code>, then <code>get_values</code> is not executed and a <code>get_last_error</code> return code of <code>RTC6_PARAM_ERROR</code> is generated.
RTC4→RTC6	<p>New command.</p> <p>Even in RTC4 Compatibility Mode, the 4 returned values are in the RTC6 20-bit range, but are transferred to the PC as 32-bit values (see comments for <code>get_value</code>).</p>
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	get_value , set_trigger , transform



Ctrl Command	get_wait_status
Function	Returns the wait state of the RTC6 board.
Call	<code>WaitStatus = get_wait_status()</code>
Result	Wait state. As an unsigned 32-bit value.
Comments	<ul style="list-style-type: none">• If list processing has been stopped at a break point ("wait marker"), then get_wait_status returns the corresponding number. See set_wait.• If no break point has been encountered, get_wait_status returns the value zero.• The list processing is resumed by calling release_wait.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_wait, release_wait



Ctrl Command	get_waveform
Function	Transfers to the PC the data that has been measured and stored onto the RTC6 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 [0...max. channel size, see set_trigger4] of measured values to be transferred. As an unsigned 32-bit value. The measured values from position 0 to (Number-1) are transferred.</p> <p>Ptr Pointer (data type <code>ULONG_PTR</code> in C and C++, an unsigned 32-bit or unsigned 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>RTC6_PARAM_ERROR</code>): <ul style="list-style-type: none"> For Number = 0. For Number > $2^{23} \times 4$ and Channel = 1 For Number > $2^{23} \times 2$ and Channel = 2 For Number > $2^{23} \times 3$ and Channel = 3 For Number > $2^{23} \times 1$ and Channel = 4 For <code>Ptr = NULL</code>. PCI transmission errors generate the get_last_error return code <code>RTC6_SEND_ERROR</code>. Before calling get_waveform, you can check by measurement_status whether a measurement session is currently running that has been 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→RTC6	Basically unchanged functionality. However: Even in RTC4 Compatibility Mode , all values are in the RTC6 20-bit range, but are transferred to the PC as 32-bit values (see comments for get_value).
RTC5→RTC6	Basically unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_trigger , set_trigger4 , get_value , get_values , get_waveform_offset



Ctrl Command	get_waveform_offset
Function	Like get_waveform , but with parameter <code>Offset</code> .
Call	<code>get_waveform_offset(Channel, Offset, Number, Ptr)</code>
Parameters	<p>Channel Measurement channel [1...4]. As an unsigned 32-bit value.</p> <p>Offset Start position. As an unsigned 32-bit value. Allowed value range: [0...max. channel size, see set_trigger4].</p> <p>Number Number of measured values to be transferred. As an unsigned 32-bit value. The measured values from position <code>Offset</code> to (<code>Offset + (Number-1)</code>) are transferred.</p> <p>Ptr Pointer (data type <code>ULONG_PTR</code> in C and C++, an unsigned 32-bit or unsigned 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"> • get_waveform is synonymous with <code>get_waveform_offset(Channel, 0, Number, Ptr)</code>. • In the following cases, no data is transferred (get_last_error return code <code>RTC6_PARAM_ERROR</code>): <ul style="list-style-type: none"> – For <code>Number = 0</code>. – For <code>Offset + Number > max. channel size</code>. – For <code>Ptr = NULL</code>. • PCI transmission errors generate the get_last_error return code <code>RTC6_SEND_ERROR</code>. • See also set_trigger/set_trigger4 (configuration of the channel).
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 610, OUT 610, RBF 614.
References	set_trigger , set_trigger4 , get_value , get_values

Ctrl Command	get_z_distance
Function	Returns the focus length value <i>l</i> for the specified point within the 3D image field .
Restriction	If the Option "3D" has not been enabled or if no 3D correction table has been assigned (see select_cor_table), then get_z_distance returns 0 and otherwise has no effect.
Call	<code>ZDistance = get_z_distance(X, Y, Z)</code>
Parameters	<p>X Absolute coordinates of the point (x y z) in the 3D image field. In bits. As a signed 32-bit value. Allowed value range: [-524,288...+524,287]. Out-of-range values are clipped to the boundary values.</p> <p>Y Like X (analogously).</p> <p>Z Like X (analogously).</p>
Result	Focus length value. [-524,288...+524,287]. As a signed 32-bit value.
Comments	<ul style="list-style-type: none"> • get_z_distance is only needed for re-calibrating the z axis in a 3-axis scan system, see Section "Checking the z axis Calibration", page 169. • The focus length value <i>l</i>: <ul style="list-style-type: none"> – Has no dimension – Corresponds to the focus length difference between the specified point (x y z) and the point (0 0 0) – Can be positive or negative • With the RTC6, ZDistance is always in 20-bit range [-524,288...+524,287]: <ul style="list-style-type: none"> – In RTC4 Compatibility Mode – In RTC5 Compatibility Mode – In RTC6 Standard Mode Important: If you do not use load_z_table_20b or load_z_table_no_20b, you must divide ZDistance by 16 and insert the result into the parabolic function $z_{out} = A + Bl + Cl^2$. • get_z_distance 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 get_z_distance has no effect and returns 0 (get_last_error return code RTC6_BUSY). • get_z_distance is not executed and returns 0 (get_last_error return code RTC6_BUSY), if: <ul style="list-style-type: none"> – the BUSY list execution status is set – the INTERNAL-BUSY list execution status is set • get_z_distance is even executed, if: <ul style="list-style-type: none"> – a list has been paused by set_wait (PAUSED list execution status set) • For 3D Image field calibration, see Chapter "3D Commands", page 237.
RTC4→RTC6	Unchanged functionality. In RTC4 Compatibility Mode , the RTC6 multiplies the values specified for X , Y and Z by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality. In RTC5 Compatibility Mode , the RTC6 multiplies the value specified for Z by 16. The allowed value range decreases accordingly.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	load_z_table , load_z_table_no , load_z_table_20b , load_z_table_no_20b



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 2D Image field .
Call	<code>goto_xy(X, Y)</code>
Parameters	<p>X Absolute x coordinate of the jump vector end point. In bits. As a signed 32-bit value. Allowed value range: [-524,288...+524,287]. Out-of-range values are clipped to the boundary values.</p> <p>Y Like X (analogously).</p>
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/ms</i>. goto_xy (unlike the list commands jump_abs and jump_rel) has no effect on the laser control signals and also does not set a Jump Delay. Previously accumulated coordinate transformations become effective by goto_xy, see list item "With <code>at_once = 2 ...</code>", page 225. goto_xy is not executed (get_last_error return code RTC6_BUSY), if: <ul style="list-style-type: none"> the BUSY list execution status is set the INTERNAL-BUSY list execution status is set an external stop signal (/STOP, /STOP2 or /Slave STOP) is present It can also be generated by automatic monitoring, see set_laser_control and range_checking. goto_xy is even executed, if: <ul style="list-style-type: none"> a list has been paused by set_wait (PAUSED list execution status set) The INTERNAL-BUSY list execution status is set while goto_xy is executed. goto_xy only returns to the user program when the movement has been completed.
RTC4→RTC6	<ul style="list-style-type: none"> Increased value range. goto_xy returns only after movement has been executed (see comments above). In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for X and Y by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 the 3D image field.						
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see select_cor_table), then goto_xyz has the same effect as goto_xy . However, split-up into Microsteps is calculated like a 3D command and hence influences the effective jump speed in the xy plane.						
Call	goto_xyz(X , Y , Z)						
Parameters	<table> <tr> <td>X</td><td>Absolute x coordinate of the jump vector end point. In bits. As a signed 32-bit value. Allowed value range: [-524,288...+524,287]. Out-of-range values are clipped to the boundary values.</td></tr> <tr> <td>Y</td><td>Like X (analogously).</td></tr> <tr> <td>Z</td><td>Like X (analogously).</td></tr> </table>	X	Absolute x coordinate of the jump vector end point. In bits. As a signed 32-bit value. Allowed value range: [-524,288...+524,287]. Out-of-range values are clipped to the boundary values.	Y	Like X (analogously).	Z	Like X (analogously).
X	Absolute x coordinate of the jump vector end point. In bits. As a signed 32-bit value. Allowed value range: [-524,288...+524,287]. Out-of-range values are clipped to the boundary values.						
Y	Like X (analogously).						
Z	Like X (analogously).						
Comments	<ul style="list-style-type: none"> Except for the additional motion in the third dimension, goto_xyz functions similarly to goto_xy (see comments there). The DirectMove3D parameter of set_delay_mode determines the type of z axis motion (linear or with stepwise correction). See also Chapter 7.3.6 "Output Values to the Scan System", page 180. 						
RTC4→RTC6	<ul style="list-style-type: none"> Increased value range. goto_xyz returns only after the motion has completed (see comments for goto_xy). RTC4 Compatibility Mode: see goto_xy. 						
RTC5→RTC6	Unchanged functionality. In RTC5 Compatibility Mode , the RTC6 multiplies the value specified for Z by 16. The allowed value range decreases accordingly.						
Version info	Available as of DLL 600, OUT 600, RBF 600.						
References	goto_xy , jump_abs_3d , jump_rel_3d						



Ctrl Command	home_position
Function	Activates the home jump mode (for the x axis and y axis) and defines the home position.
Call	<code>home_position(XHome, YHome)</code>
Parameters	<p><code>XHome</code> Absolute x coordinate of the home position. In bits. As a signed 32-bit value. Allowed value range: [-524,288...+524,287]. Larger values are clipped.</p> <p><code>YHome</code> Like <code>XHome</code> (analogously).</p>
Comments	<ul style="list-style-type: none"> • home_position defines the coordinates of a home jump to be executed, for example, upon reaching the end of a list. Accordingly, a home return to the last valid position is then executed again at start. The home jump and home return are executed at jump speed. • home_position is intended for laser systems that do not allow fast switching of the laser. After calling home_position, 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 list execution status is set. • A beam dump should be placed in the home position. • The home jump mode is deactivated by <code>home_position(0, 0)</code>, even if it has been activated by home_position_xyz.
<code>RTC4→RTC6</code>	Unchanged functionality. In RTC4 Compatibility Mode , the RTC6 multiplies the specified values for <code>XHome</code> and <code>YHome</code> by 16. The allowed value range decreases accordingly.
<code>RTC5→RTC6</code>	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	home_position_xyz



Ctrl Command	home_position_xyz						
Function	Activates the home jump mode (for the x, y and z axis) and defines the home position.						
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see select_cor_table), then home_position_xyz has the same effect as home_position . However, split-up into Microsteps 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	<table> <tr> <td>XHome</td> <td>Absolute x coordinate of the home position. In bits. As a signed 32-bit value. Allowed value range: [-524,288...+524,287]. Out-of-range values are clipped to the boundary values.</td> </tr> <tr> <td>YHome</td> <td>Like XHome (analogously).</td> </tr> <tr> <td>ZHome</td> <td>Like XHome (analogously).</td> </tr> </table>	XHome	Absolute x coordinate of the home position. In bits. As a signed 32-bit value. Allowed value range: [-524,288...+524,287]. Out-of-range values are clipped to the boundary values.	YHome	Like XHome (analogously).	ZHome	Like XHome (analogously).
XHome	Absolute x coordinate of the home position. In bits. As a signed 32-bit value. Allowed value range: [-524,288...+524,287]. Out-of-range values are clipped to the boundary values.						
YHome	Like XHome (analogously).						
ZHome	Like XHome (analogously).						
Comments	<ul style="list-style-type: none"> Except for the additional motion in the third dimension, home_position_xyz functions similarly to home_position (see comments there). The home jump mode is deactivated by: <ul style="list-style-type: none"> – <code>home_position(0, 0)</code> – <code>home_position_xyz(0, 0, 0)</code> The DirectMove3D parameter of set_delay_mode determines the type of z axis motion (linear or with stepwise correction). 						
RTC4→RTC6	<p>New command.</p> <p>In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for XHome, YHome and ZHome by 16. The allowed value range decreases accordingly.</p>						
RTC5→RTC6	<p>Unchanged functionality.</p> <p>In RTC5 Compatibility Mode, the RTC6 multiplies the specified value for ZHome by 16. The allowed value range decreases accordingly.</p>						
Version info	Available as of DLL 600, OUT 600, RBF 600.						
References	home_position						

Undelayed Short List Command	if_cond
Function	<p><i>Conditional command execution:</i> if_cond 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>(= if the bits specified in Mask1 are 1 and the bits specified in Mask0 are 0). Otherwise, this list command is skipped.</p>
Call	<code>if_cond(Mask1, Mask0)</code>
Parameters	<p>Mask1 16-bit mask. As an unsigned 32-bit value. Only the lower 16 bits are evaluated.</p> <p>Mask0 See Mask1.</p>
Comments	<ul style="list-style-type: none"> • See also Chapter 9.3.2 "Conditional Command Execution", page 294.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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> if_fly_x_overflow immediately executes the directly subsequent list command, if the condition in accordance with <code>Mode</code> for the x axis has been fulfilled. Otherwise, this list command is skipped.</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 Bit #4 = 1 or error bit Bit #5 = 1). > 0: An overflow occurred (error bit Bit #5 = 1). < 0: An underflow occurred (error bit Bit #4 = 1).</p>
Comments	<ul style="list-style-type: none"> • For usage of if_fly_x_overflow, see Section "Customer-Defined Monitoring Area", page 255. • The error bits queried in accordance with <code>Mode</code> (error bit Bit #4 and/or error bit Bit #5) are reset.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	get_marking_info , set_fly_limits , clear_fly_overflow , clear_fly_overflow_ctrl , 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> if_fly_y_overflow immediately executes the directly subsequent list command, if the condition in accordance with <code>Mode</code> for the y axis has been fulfilled. Otherwise, this list command is skipped.
Call	<code>if_fly_y_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 Bit #6 = 1 or error bit Bit #7 = 1). > 0: An overflow occurred (error bit Bit #7 = 1). < 0: An underflow occurred (error bit Bit #6 = 1).
Comments	<ul style="list-style-type: none"> For usage of if_fly_y_overflow, see Section "Customer-Defined Monitoring Area", page 255. The error bits queried in accordance with <code>Mode</code> (error bit Bit #6 and/or error bit Bit #7) are reset.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	get_marking_info , set_fly_limits , clear_fly_overflow , clear_fly_overflow_ctrl , 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> if_fly_z_overflow immediately executes the directly subsequent list command, if the condition in accordance with <code>Mode</code> for the z axis has been fulfilled. Otherwise, this list command is skipped.
Call	<code>if_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 <code>Bit #24</code> = 1 or error bit <code>Bit #25</code> = 1). > 0: An overflow occurred (error bit <code>Bit #25</code> = 1). < 0: An underflow occurred (error bit <code>Bit #24</code> = 1).
Comments	<ul style="list-style-type: none"> For usage of <code>if_fly_z_overflow</code>, see also Chapter 8.6.9 "Monitoring Processing-on-the-fly Corrections", page 254. The error bits queried in accordance with <code>Mode</code> (error bit <code>Bit #24</code> and/or error bit <code>Bit #25</code>) are reset.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	get_marking_info , set_fly_limits_z , clear_fly_overflow , clear_fly_overflow_ctrl , 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/activate_fly_xy_encoder or activate_fly_2d/activate_fly_2d_encoder : If the error bit is set, then the next list command is executed 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/activate_fly_2d_encoder and activate_fly_xy/activate_fly_xy_encoder. 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/activate_fly_xy_encoder or activate_fly_2d/activate_fly_2d_encoder. If you still need it for subsequent querying by <code>get_marking_info(Bit #9)</code>, then you can include a renewed call of activate_fly_2d/activate_fly_2d_encoder 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/activate_fly_2d_encoder or activate_fly_xy/activate_fly_xy_encoder does not reset any already-set error bit. It remains set for get_marking_info until get_marking_info has been called.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	activate_fly_2d , activate_fly_2d_encoder , activate_fly_xy , activate_fly_xy_encoder , get_marking_info



Undelayed Short List Command	if_not_cond
Function	<p><i>Conditional command execution:</i> if_not_cond immediately executes the directly following list command, if the current IOvalue at the EXTENSION 1 socket connector's 16-bit digital input port <i>does not meet the following condition</i>:</p> $((\text{IOvalue AND Mask1}) = \text{Mask1}) \text{ AND } (((\text{not IOvalue}) \text{ AND Mask0}) = \text{Mask0})$ <p>(= if the bits specified in Mask1 are <i>not 1</i> or the bits specified in Mask0 are <i>not 0</i>). Otherwise, this list command is skipped.</p>
Call	<code>if_not_cond(Mask1, Mask0)</code>
Parameters	<p>Mask1 16-bit mask. As an unsigned 32-bit value. Only the lower 16 bits are evaluated.</p> <p>Mask0 See Mask1.</p>
Comments	<ul style="list-style-type: none"> • See also Chapter 9.3.2 "Conditional Command Execution", page 294.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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> if_not_fly_x_overflow immediately executes the directly subsequent list command, if the condition in accordance with <code>Mode</code> for the x axis is <i>not</i> fulfilled. Otherwise, this list command is skipped.
Call	<code>if_not_fly_x_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 Bit #4 = 1 or error bit Bit #5 = 1). > 0: An overflow occurred (error bit Bit #5 = 1). < 0: An underflow occurred (error bit Bit #4 = 1).
Comments	<ul style="list-style-type: none"> For usage of if_not_fly_x_overflow, see Section "Customer-Defined Monitoring Area", page 255. The error bits queried in accordance with <code>Mode</code> (error bit Bit #4 and/or error bit Bit #5) are reset.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	get_marking_info , set_fly_limits , clear_fly_overflow , clear_fly_overflow_ctrl , 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> if_not_fly_y_overflow immediately executes the directly subsequent list command, if the condition in accordance with <code>Mode</code> for the y axis is <i>not</i> fulfilled. Otherwise, this list command is skipped.
Call	<code>if_not_fly_y_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 Bit #6 = 1 or error bit Bit #7 = 1). > 0: An overflow occurred (error bit Bit #7 = 1). < 0: An underflow occurred (error bit Bit #6 = 1).
Comments	<ul style="list-style-type: none"> For usage of if_not_fly_y_overflow, see Section "Customer-Defined Monitoring Area", page 255. The error bits queried in accordance with <code>Mode</code> (error bit Bit #6 and/or error bit Bit #7) are reset.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	get_marking_info , set_fly_limits , clear_fly_overflow , clear_fly_overflow_ctrl , 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> if_not_fly_z_overflow immediately executes the directly subsequent list command, if the condition in accordance with Mode for the z axis has not been fulfilled. Otherwise, this list command is skipped.
Call	<code>if_not_fly_z_overflow(Mode)</code>
Parameters	Mode To-be-evaluated condition. As a signed 32-bit value. = 0: Some kind of boundary exceedance occurred (error bit Bit #24 = 1 or error bit Bit #25 = 1). > 0: An overflow occurred (error bit Bit #25 = 1). < 0: An underflow occurred (error bit Bit #24 = 1).
Comments	<ul style="list-style-type: none"> For usage of if_not_fly_z_overflow, see also Chapter 8.6.9 "Monitoring Processing-on-the-fly Corrections", page 254. The error bits queried in accordance with Mode (error bit Bit #24 and/or error bit Bit #25) are reset.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	get_marking_info , set_fly_limits_z , clear_fly_overflow , clear_fly_overflow_ctrl , if_fly_z_overflow

Undelayed Short List Command	if_not_pin_cond
Function	<i>Conditional command execution:</i> if_not_pin_cond immediately executes the directly following list command, if the current IOvalue at the LASER Connector 's 2-bit digital input port does not meet the following condition: $((\text{IOvalue AND Mask1}) = \text{Mask1}) \text{ AND } (((\text{not IOvalue}) \text{ AND Mask0}) = \text{Mask0})$ (= if the bits specified in Mask1 are <i>not 1</i> or the bits specified in Mask0 are <i>not 0</i>). Otherwise, this list command is skipped.
Call	<code>if_not_pin_cond(Mask1, Mask0)</code>
Parameters	Mask1 2-bit mask. As an unsigned 32-bit value. Only the lower 2 bits are evaluated. Mask0 See Mask1 .
Comments	<ul style="list-style-type: none"> See also Chapter 9.3.2 "Conditional Command Execution", page 294.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	if_pin_cond , if_cond , if_not_cond



Undelayed Short List Command	if_pin_cond
Function	<p><i>Conditional command execution:</i></p> <p>if_pin_cond 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> $((\text{IOvalue AND Mask1}) = \text{Mask1}) \text{ AND } (((\text{not IOvalue}) \text{ AND Mask0}) = \text{Mask0})$ <p>(= if the bits specified in Mask1 are 1 and the bits specified in Mask0 are 0). Otherwise, this list command is skipped.</p>
Call	<code>if_pin_cond(Mask1, Mask0)</code>
Parameters	<p>Mask1 2-bit mask. As an unsigned 32-bit value. Only the lower 2 bits are evaluated.</p> <p>Mask0 See Mask1.</p>
Comments	<ul style="list-style-type: none"> • See also Chapter 9.3.2 "Conditional Command Execution", page 294.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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, No)</code>
Parameters	<p>OffsetX Reference value of the x axis encoder. As a signed 32-bit value. Allowed value range: depends on the compensation table loaded with load_fly_2d_table. Default values (after load_program_file): = 0.</p> <p>OffsetY Like OffsetX (analogously).</p> <p>No Number of the 2D compensation table to be used. See also page 249.</p>
Comments	<ul style="list-style-type: none"> The final encoder value for the 2D correction value + reference value must not exceed the allowed range. Otherwise, clipping occurs (see also load_fly_2d_table). No = 1: use table 1. No = 2: use table 2. No = 0: use the table that has been used up to now No > 2: do not use a table. The tables to be used must have been validly loaded by load_fly_2d_table beforehand. With No > 2, the table remains validly loaded. By load_fly_2d_table(Name = NULL), the table becomes invalid.
RTC4→RTC6	New command.
RTC5→RTC6	Changed functionality. <ul style="list-style-type: none"> Parameter No.
Version info	Available as of DLL 614, OUT 614, RBF 619.
References	set_fly_2d , load_fly_2d_table , get_fly_2d_offset

Ctrl Command	init_rtc6_dll
Function	Initializes control of the installed RTC6 boards for a user program.
Call	<code>InitErrorNo = init_rtc6_dll()</code>
Result	<p>Error code. As an unsigned 32-bit value. If multiple errors occurred simultaneously, then multiple bits are set. The assignment between bit numbers, error types and error constants is identical to those for get_error.</p>
Comments	<ul style="list-style-type: none"> • <code>init_rtc6_dll</code> must be called by each user program at the beginning, so that an RTC6 board can be addressed by the user program at all. • <code>init_rtc6_dll</code> searches for all present RTC6 PCIe Boards (not for RTC6 Ethernet Boards! – see Chapter 16.5.2 "About Searching RTC6 Ethernet Boards", page 886 on this topic) and establishes the RTC6 board management. If no RTC6 board is found (for example, when only RTC6 Ethernet Boards are operated), then error code <code>RTC6_NO_PCIE_CARD_FOUND</code> is returned (see get_error). • <code>init_rtc6_dll</code> automatically assigns the user program access rights to the found boards (as by acquire_rtc), 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>RTC6_ACCESS_DENIED</code> error code by <code>init_rtc6_dll</code>. The boards are then only accessible by these applications through RTC6 DLL-internal functions that require no access rights – for example, get_error, get_last_error or select_rtc (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 release_rtc or free_rtc6_dll before the board can be used by another user program by <code>init_rtc6_dll</code> or acquire_rtc. An <code>RTC6_ACCESS_DENIED</code> error code is returned if access has been denied by at least one of the found boards. Which board(s) denied access can be determined by <code>n_get_error(CardNo)</code> (<code>CardNo</code> from 1 to the number of found boards) or directly after <code>init_rtc6_dll</code> (before the next command) by <code>n_get_last_error(CardNo)</code>. • If a board is acquired by <code>init_rtc6_dll</code> (as by acquire_rtc), then a version compatibility check is performed. If a version error is detected, then access to the board is denied (return code <code>RTC6_ACCESS_DENIED RTC6_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.

Ctrl Command	init_RTC6_dll
Comments (cont'd)	<ul style="list-style-type: none"> If a user program calls <code>init_RTC6_dll</code> multiple times, then the RTC6 board management created by the prior call is deleted for this user program and the originally-assigned access rights canceled. The RTC6 board management is then newly created and access rights are newly granted again. Each initialization of a user program by <code>init_RTC6_dll</code> causes the RTC6 DLL-internal numbers for all found RTC6 boards to be newly reassigned. The relationship between the RTC6 DLL-internal numbers and the installed boards can be determined by <code>get_serial_number</code>. When the accessible board with the smallest RTC6 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 RTC6 DLL-internal number (see <code>rtc6_count_cards</code>) is the active board, in which case only RTC6 DLL-internal commands that require no access rights can be used. Also observe Chapter 6.7.1 "Notes on Board Acquisition by a User Program", page 127. <code>init_RTC6_dll</code> is available even without explicit access rights to any particular RTC6 board. <code>init_RTC6_dll</code> is not available as a multi-board command. After initialization of the RTC6 DLL with <code>init_RTC6_dll</code>, the RTC6 Standard Mode is set by default. After that, a different RTC6 DLL operational mode can be set, see <code>set_RTC4_mode</code> and <code>set_RTC5_mode</code>. <code>init_RTC6_dll</code> does not trigger an initialization of RTC6 boards. Only <code>load_program_file</code> performs an initialization of RTC6 boards.
RTC4→RTC6	New command.
RTC5→RTC6	New command. The functionalities of <code>init_RTC6_dll</code> and the RTC5 command <code>init_RTC5_dll</code> are identical.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	<code>set_RTC4_mode</code> , <code>set_RTC5_mode</code>

Normal List Command	jump_abs
Function	Moves the output point (for the laser focus) along a 2D vector at jump speed from the current position to the specified position (absolute coordinate values) within a 2D Image field .
Call	<code>jump_abs(X, Y)</code>
Parameters	<p>X Absolute x coordinate of the jump vector end point. In bits. As a signed 32-bit value. Allowed value range: [-268,435,456...+268,435,455]. Out-of-range values are clipped to the boundary values. The complete value range is only usable as a virtual Image field for example, for (enabled) Processing-on-the-fly applications.</p> <p>Y Like X (analogously).</p>
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 bits/ms. The Signals for "Laser Active" Operation are switched off before the jump and remain off during 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, jump_abs itself still requires a 10 µs clock cycle for execution. Previously accumulated coordinate transformations become effective by jump_abs, see list item "With at_once = 2 ...", page 225.
RTC4→RTC6	Unchanged functionality. In addition: increased value range. In RTC4 Compatibility Mode , the RTC6 multiplies the values specified for X and Y by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality. In addition: increased value range.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 the 3D image field .
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see select_cor_table), then jump_abs_3d has the same effect as jump_abs . However, split-up into Microsteps 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	<p>X Absolute x coordinate of the jump vector end point. In bits. As a signed 32-bit value. Allowed value range: [-268,435,456...+268,435,455]. Out-of-range values are clipped to the boundary values. The complete value range is only usable as a virtual Image field for example, for (enabled) Processing-on-the-fly applications.</p> <p>Y Like X (analogously).</p> <p>Z Like X (analogously), except Allowed value range: [-524,288...+524,287].</p>
Comments	<ul style="list-style-type: none"> Except for the additional motion in the third dimension, jump_abs_3d functions similarly to jump_abs (see comments there). The DirectMove3D parameter of set_delay_mode determines the type of z axis motion (linear or with stepwise correction).
RTC4→RTC6	Unchanged functionality. In addition: increased value range. In RTC4 Compatibility Mode , the RTC6 multiplies the values specified for X , Y and Z by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality. In addition: increased value range. In RTC5 Compatibility Mode , the RTC6 multiplies the value specified for Z by 16. The allowed value range decreases accordingly.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	jump_abs , jump_rel_3d , timed_jump_abs_3d

Normal List Command	jump_abs_drill
Call	<code>jump_abs_drill(X, Y, DrillTime)</code>
Comments	<ul style="list-style-type: none"> This command is described, for example, in the manual for the intelliDRILL de II 20 Module.

Normal List Command	jump_abs_drill_2
Call	<code>jump_abs_drill_2(X, Y, DrillTime, XOff, YOff)</code>
Comments	<ul style="list-style-type: none"> This command is described, for example, in the manual for the intelliDRILL de II 20 Module.



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 2D Image field .
Call	<code>jump_rel(dx, dy)</code>
Parameters	<p><code>dx</code> Relative x coordinate of the jump vector end point. In bits. As a signed 32-bit value. Allowed value range: [-268,435,456...+268,435,455]. Out-of-range values are clipped to the boundary values. The complete value range is only usable as a virtual Image field, for example, for (enabled) Processing-on-the-fly applications.</p> <p><code>dy</code> Like <code>dx</code> (analogously).</p>
Comments	<ul style="list-style-type: none"> The coordinates for the jump vector end point are to be supplied as relative coordinates with respect to the current position. Otherwise, jump_rel is identical to jump_abs (see comments there).
RTC4→RTC6	Unchanged functionality. In addition: increased value range. In RTC4 Compatibility Mode , the RTC6 multiplies the specified values for <code>dx</code> and <code>dy</code> by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality. In addition: increased value range.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 the 3D image field .
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see select_cor_table), then jump_rel_3d has the same effect as jump_rel . However, split-up into Microsteps 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>dx Relative x coordinate of the jump vector end point. In bits. As a signed 32-bit value. Allowed value range: [-268,435,456...+268,435,455]. Out-of-range values are clipped to the boundary values. The complete value range is only usable as a virtual Image field for example, for (enabled) Processing-on-the-fly applications.</p> <p>dy Like dx (analogously).</p> <p>dz Like dx (analogously), except Allowed value range: [-524,288...+524,287].</p>
Comments	<ul style="list-style-type: none"> The coordinates for the jump vector end point are to be supplied as relative coordinates with respect to the current position. Otherwise, jump_rel_3d is identical to jump_abs_3d (see comments there).
RTC4→RTC6	Unchanged functionality. In addition: increased value range. In RTC4 Compatibility Mode , the RTC6 multiplies the specified values for dx , dy and dz by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality. In addition: increased value range.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	jump_abs_3d , jump_abs , jump_rel , timed_jump_rel_3d

Normal List Command	jump_rel_drill
Call	<code>jump_rel_drill(dx, dy, DrillTime)</code>
Comments	<ul style="list-style-type: none"> This command is described, for example, in the manual for the intelliDRILL de II 20 Module.

Normal List Command	jump_rel_drill_2
Call	<code>jump_rel_drill_2(dx, dy, DrillTime, XOff, YOff)</code>
Comments	<ul style="list-style-type: none"> This command is described, for example, in the manual for the intelliDRILL de II 20 Module.

Variable List Command	laser_on_list
Function	Turns on the Signals for "Laser Active" Operation for a specified time interval.
Call	<code>laser_on_list(Period)</code>
Parameters	<p>Period Time interval. In bits. As an unsigned 32-bit value. 1 bit equals $10\ \mu\text{s}$. Allowed value range: $0 \leq \text{Period} \leq (2^{31}-1)$.</p>
Comments	<ul style="list-style-type: none"> Bit #31 of Period is ignored (see also Comments at laser_on_pulses_list). The 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 183. 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 (for example, 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 (for example, jump_abs or list_nop). The currently set LaserOff Delay is applied. laser_on_list is useful for marking single dots, see Chapter 7.1.3 "Marking Single Dots", page 140. Wobbel Mode, see Chapter 8.4 "Wobbel Mode", page 231, is retained but ignored. For <code>Period = 0</code> the laser control signals do not turn on. laser_on_list is a short list command then. 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→RTC6	Unchanged functionality. In addition: increased value range.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 <code>Pulses > 65,535</code> the function of <code>laser_on_pulses_list</code> is identical to <code>laser_on_list</code> .				
Call	<code>laser_on_pulses_list(Period, Pulses)</code>				
Parameters	<table> <tr> <td>Period</td> <td>Time interval. In bits. As an unsigned 32-bit value. 1 bit equals 10 μs. Allowed value 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 value range: $0 \leq \text{Pulses} \leq 65,535$ or larger (see comments below).</td> </tr> </table>	Period	Time interval. In bits. As an unsigned 32-bit value. 1 bit equals 10 μ s. Allowed value range: $0 \leq \text{Period} \leq (2^{32}-1)$.	Pulses	Number of external signal pulses. As an unsigned 32-bit value. Allowed value range: $0 \leq \text{Pulses} \leq 65,535$ or larger (see comments below).
Period	Time interval. In bits. As an unsigned 32-bit value. 1 bit equals 10 μ s. Allowed value range: $0 \leq \text{Period} \leq (2^{32}-1)$.				
Pulses	Number of external signal pulses. As an unsigned 32-bit value. Allowed value range: $0 \leq \text{Pulses} \leq 65,535$ or larger (see comments below).				
Comments	<ul style="list-style-type: none"> • <code>laser_on_pulses_list</code> is useful for marking single dots in Laser Mode 6, 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 port (see Section "2-Bit Digital Input Port", page 73). By <code>set_laser_control(Bit #5)</code>, you can specify whether signal pulses should be counted at rising or falling edges. • If <code>Period = 0</code>, then <code>laser_on_pulses_list</code> has no effect. Then <code>laser_on_pulses_list</code> is a short list command. • If $0 < \text{Period} \leq (2^{31}-1)$, then <code>laser_on_pulses_list</code> duration is always <code>Period</code> clocks (that is, <code>Period</code> \times 10 μs), even if the specified number of external signal pulses expire in a shorter time interval. • If $2^{31} \leq \text{Period} \leq (2^{32}-1)$, <code>laser_on_pulses_list</code> maximum duration is $(\text{Period} - 2^{31})$ clocks $((\text{Period} - 2^{31}) \times 10 \mu\text{s})$. Here, however, <code>laser_on_pulses_list</code> terminates as soon as the specified number of external signal pulses has been detected. • If <code>Pulses > 65,535</code>, then <code>laser_on_pulses_list</code>'s function is identical to <code>laser_on_list</code> (external signal pulses are not taken into account, see comments there). Otherwise (for $0 \leq \text{Pulses} \leq 65,535$), <code>laser_on_pulses_list</code> (in contrast to <code>laser_on_list</code>) do not toggle the laser control signals between "laser active" and "laser standby" operation, but instead only switches the LASERON signal, 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 $1 \leq \text{Pulses} \leq 65,535$, then the LASERON signal does switch on upon the edge (according the polarity set by <code>set_laser_control(Bit #5)</code>) 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 port 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 <code>Pulses</code> = 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 183. • While laser_on_pulses_list is executed, the set position of the scanners is not changed. The next list command is executed when the programmed time interval <code>Period</code> has passed. • The Wobbel Mode, see Chapter 8.4 "Wobbel Mode", page 231, is retained but ignored. • Any softstarts that were defined are not executed.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	laser_on_list , para_laser_on_pulses_list



Ctrl Command	laser_signal_off
Function	Turns off the Signals for "Laser Active" Operation immediately.
Call	<code>laser_signal_off()</code>
Comments	<ul style="list-style-type: none"> • laser_signal_off is intended for direct laser control in combination with laser_signal_on. • laser_signal_off is not executed (get_last_error return code RTC6_BUSY), if: <ul style="list-style-type: none"> – the BUSY list execution status is set • laser_signal_off is even executed, if: <ul style="list-style-type: none"> – a list has been paused by set_wait (PAUSED list execution status set) – the INTERNAL-BUSY list execution status is set
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	laser_signal_on

Normal List Command	laser_signal_off_list
Function	Like laser_signal_off , but a list command.
Call	<code>laser_signal_off_list()</code>
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	laser_signal_off



Ctrl Command	laser_signal_on
Function	Turns on the Signals for "Laser Active" Operation immediately.
Call	<code>laser_signal_on()</code>
Comments	<ul style="list-style-type: none"> • <i>Caution! Check the beam path before turning on the laser!</i> • The Signals for "Laser Active" Operation must first be selected by set_laser_mode, defined by further commands, and enabled by set_laser_control or enable_laser before they can be switched on with laser_signal_on, see Chapter 7.4 "Laser Control", page 183. • laser_signal_on is intended for turning on the laser directly, for example, for alignment purposes. • The Signals for "Laser Active" Operation must be turned off by laser_signal_off. • laser_signal_on is not executed (get_last_error return code RTC6_BUSY), if: <ul style="list-style-type: none"> – the BUSY list execution status is set – the INTERNAL-BUSY list execution status is set • laser_signal_on is even executed, if: <ul style="list-style-type: none"> – a list has been paused by set_wait (PAUSED list execution status set)
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	laser_signal_off

Normal List Command	laser_signal_on_list
Function	Like laser_signal_on , but a list command and never ignored.
Call	<code>laser_signal_on_list()</code>
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	laser_signal_on

Undelayed Short List Command	list_call
Function	Causes an unconditional jump to a subroutine that starts at the specified absolute address (in any desired location within list memory).
Call	<code>list_call(Pos)</code>
Parameters	<p>Pos Absolute jump address $[0\dots(2^{23}-1)]$. As an unsigned 32-bit value.</p>
Comments	<ul style="list-style-type: none"> The first command of a subroutine called by list_call is executed (possibly after a list_continue) 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 111. Each subroutine must be terminated by list_return so that after the subroutine (including the terminating list_return) has been processed, execution continues with the command that follows the subroutine-call command. This, too, executes (after a possible list_continue) immediately and without delay. If set_end_of_list is encountered instead of the expected list_return, 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 mark_text does not execute to completion. If the end of a list memory area ("List 1" or "List 2") is reached without having encountered a list_return or set_end_of_list, then execution continues at the start of the current list. If such a situation occurs in the protected list memory area "List 3", then a compulsory list_return command is inserted and executed. The list_call command is replaced by a list_nop if <code>Pos > (2²³-1)</code> or if <code>Pos</code> is also the current address (get_last_error return code <code>RTC6_PARAM_ERROR</code>). If a called subroutine executes a further list_call to the address of the calling list_call command (recursive call), then the resulting endless loop is terminated as soon as the 63-nested-call upper limit is reached. Further list_call commands are then ignored and the next command is instead executed. If the subroutine starts directly at the address which follows list_call, then the subroutine is executed once again after list_return (see also comments on a missing corresponding function call in the list_return command description). The next processed command is the one which follows after list_return (see also list_repeat...list_until). This bypasses the possibly unwanted list processing.



Undelayed Short List Command	list_call
Comments (cont'd)	<ul style="list-style-type: none"> The BUSY list status readable by read_status is altered by list_call if the called address is in the list memory area ("List 1" or "List 2"). If this address is instead in protected list memory area "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, get_status too returns the calling location's List Execution Status, and get_out_pointer returns the position of the calling list_call as the output pointer's position. Absolute Vector commands 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[*] Commands, "Arc" commands 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 has been previously enabled – execution continues at the start of the same list after processing of the called subroutine. list_call(Pos) is synonymous with list_call_repeat(Pos, 1).
RTC4→RTC6	Basically unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 address (in any desired area of list memory). In the called subroutine, any absolute Vector commands and "Arc" commands receive an offset (based on the current coordinates at the time of the call).
Call	<code>list_call_abs(Pos)</code>
Parameters	Pos Absolute jump address [0...(2 ²³ -1)]. As an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> • list_call_abs is basically identical to list_call (see comments there). However, list_call_abs results in a different execution of absolute Vector commands 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 commands 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 commands 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 synonymous with list_call_abs_repeat(Pos, 1).
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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> list_call_abs_cond executes list_call_abs(<i>Pos</i>), if the current <i>IOvalue</i> 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>(= if the bits specified in <i>Mask1</i> are 1 and the bits specified in <i>Mask0</i> are 0). Otherwise, the directly following list command is immediately executed.</p>
Call	<code>list_call_abs_cond(Mask1, Mask0, Pos)</code>
Parameters	<p>Mask1 16-bit mask. As an unsigned 32-bit value. Only the least significant 16 bits are evaluated.</p> <p>Mask0 See <i>Mask1</i>.</p> <p>Pos Absolute jump address [0...(2²³-1)]. As an unsigned 32-bit value.</p>
Comments	<ul style="list-style-type: none"> • See list_call_abs. • See also Chapter 9.3.2 "Conditional Command Execution", page 294.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 address (in any desired area of list memory) and executes its body several times.
Call	<code>list_call_abs_repeat(Pos, Number)</code>
Parameters	<p>Pos Absolute jump address [0...(2²³-1)]. As with list_call_abs: As an unsigned 32-bit value.</p> <p>Number Number of repetitions. As an unsigned 32-bit value. Number = 0 is treated as Number = 1.</p>
Comments	<ul style="list-style-type: none"> • list_call_abs(<i>Pos</i>) is synonymous with <code>list_call_abs_repeat(Pos, 1)</code>. • list_call_abs_repeat avoids an empty cycle at the repetition, which otherwise inevitably occurs with list_call_abs...list_call_abs or list_repeat...list_call_abs...list_until constructions. • See list_call_repeat and list_call_abs.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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	<i>Conditional subroutine call: list_call_abs_cond</i> executes list_call (<i>Pos</i>), if the current <i>IOvalue</i> at the EXTENSION 1 socket connector's 16-bit digital input port meets the following condition: $((\text{IOvalue AND Mask1}) = \text{Mask1}) \text{ AND } (((\text{not IOvalue}) \text{ AND Mask0}) = \text{Mask0})$ (= if the bits specified in <i>Mask1</i> are 1 and the bits specified in <i>Mask0</i> are 0). Otherwise, the directly following list command is immediately executed.
Call	<code>list_call_cond(Mask1, Mask0, Pos)</code>
Parameters	<p><i>Mask1</i> 16-bit mask. As an unsigned 32-bit value. Only the lower 16 bits are evaluated.</p> <p><i>Mask0</i> See <i>Mask1</i>.</p> <p><i>Pos</i> Absolute jump address $[0\dots(2^{23}-1)]$. As an unsigned 32-bit value.</p>
Comments	<ul style="list-style-type: none"> • See list_call. • See also Chapter 9.3.2 "Conditional Command Execution", page 294.
RTC4→RTC6	Basically unchanged functionality (see list_call).
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	list_call

Undelayed Short List Command	list_call_repeat				
Function	Causes an unconditional jump to a subroutine that starts at the specified absolute address (in any desired area of list memory) and executes its body several times.				
Call	<code>list_call_repeat(Pos, Number)</code>				
Parameters	<table> <tr> <td>Pos</td> <td>Absolute jump address [0...(2²³-1)]. As with list_call: As an unsigned 32-bit value.</td> </tr> <tr> <td>Number</td> <td>Number of repetitions. As an unsigned 32-bit value. Number = 0 is treated as Number = 1.</td> </tr> </table>	Pos	Absolute jump address [0...(2 ²³ -1)]. As with list_call : As an unsigned 32-bit value.	Number	Number of repetitions. As an unsigned 32-bit value. Number = 0 is treated as Number = 1.
Pos	Absolute jump address [0...(2 ²³ -1)]. As with list_call : As an unsigned 32-bit value.				
Number	Number of repetitions. As an unsigned 32-bit value. Number = 0 is treated as Number = 1.				
Comments	<ul style="list-style-type: none"> • list_call(Pos) is synonymous with list_call_repeat(Pos, 1). • list_call_repeat avoids an empty cycle at the repetition, which otherwise inevitably occurs with list_call...list_call or list_repeat...list_call...list_until constructions. • By list_call_repeat, for example, trajectories (see Glossary entry on page 29) 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 list_call_repeat another subroutine call follows immediately (nested calls). This can be avoided, if there is for example, at least one micro_vector[*] 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 list_call_repeat. Thus, the complete list memory can continuously be used for list commands without reserving a special area for protected subroutines. • After a list_return the next processed command is the one which follows directly after list_call_repeat. However, if the subroutine start follows directly after list_call_repeat, then the next processed command is the one which follows directly after list_return. 				
RTC4→RTC6	New command.				
RTC5→RTC6	Unchanged functionality.				
Version info	Available as of DLL 600, OUT 600, RBF 600.				
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 memory.
Call	<code>list_continue()</code>
Comments	<ul style="list-style-type: none"> Execution of list_continue (as does list_nop) requires 10 μs. When list_continue 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 cycle (the short list command's "effective" execution time is then 10 μs). Unlike list_nop, however, list_continue neither modifies laser control 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 cycle without interrupting Polylines by switching off the laser. With exceedance of the maximum allowed number of directly consecutive short list commands, an empty cycle (which exactly corresponds list_continue) is automatically inserted. You can also use list_continue to separate from each other several outputs to the same output port. Without list_continue, for example, multiple directly consecutive write_da_x_list calls (short list commands) would occur in a single 10 μs clock cycle, whereby some values to the analog output port would never get outputted. The RTC6 never uses list_continue as a placeholder for other (rejected) list commands, but instead always uses list_nop.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	list_nop



Undelayed Short List Command	list_jump_cond
Function	<p><i>Conditional absolute list jump:</i> list_jump_cond executes 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>(= 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	list_jump_cond(Mask1, Mask0, Pos)
Parameters	<p>Mask1 16-bit mask. As an unsigned 32-bit value. Only the lower 16 bits are evaluated.</p> <p>Mask0 See Mask1.</p> <p>Pos Absolute jump address $[0\dots(2^{23}-1)]$. As an unsigned 32-bit value.</p>
Comments	<ul style="list-style-type: none"> • list_jump_cond is synonymous with list_jump_pos_cond (see comments there).
RTC4→RTC6	<p>Basically unchanged functionality. However:</p> <ul style="list-style-type: none"> • Jumps into or out from the protected list memory area "List 3" are not allowed (illegal Jump commands are ignored during processing, see also list_jump_pos).
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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>
Parameters	<code>Pos</code> Absolute jump address [0...(2 ²³ -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 list memory area "List 3" are not allowed. Illegal Jump commands are transmitted unaltered to the RTC6, 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 119). 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 Stop). Decisive are the runtime conditions. When reconfiguring list memory or converting a subroutine, an originally valid jump address might become invalid 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 Chapter 6.4.2 "List Status", page 106). 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). list_jump_pos is synonymous with set_list_jump.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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> list_jump_pos_cond executes list_jump_pos(<i>Pos</i>), if the current <i>IOvalue</i> 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>(= if the bits specified in <i>Mask1</i> are 1 and the bits specified in <i>Mask0</i> are 0). Otherwise, the directly following list command is immediately executed.</p>
Call	<code>list_jump_pos_cond(Mask1, Mask0, Pos)</code>
Parameters	<p>Mask1 16-bit mask. As an unsigned 32-bit value. Only the lower 16 bits are evaluated.</p> <p>Mask0 See <i>Mask1</i>.</p> <p>Pos Absolute jump address [0...(2^{23}–1)]. As an unsigned 32-bit value.</p>
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 (<i>Pos</i> = list position of the Jump command), for example, to wait for confirmation of a signal. • list_jump_pos_cond is synonymous with list_jump_cond. • See also Chapter 9.3.2 "Conditional Command Execution", page 294.
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_pos_cond(0, \$0008, get_input_pointer);</code> • skip the next two list commands if the state of the input port is xxxx xxxx xxxx 0110: <code>list_jump_pos_cond(6, 9, get_input_pointer + 3);</code> • See also Section "Example Code (Pascal)", page 295.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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)
Parameters	Pos Jump distance [(-2 ²³ +1)...(2 ²³ -1)]. As a signed 32-bit value.
Comments	<ul style="list-style-type: none"> • list_jump_rel enables implementation of branching (for example, "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 list input pointer can be queried by get_list_pointer. • list_jump_rel is usable in all list areas, including the protected list memory "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 list_jump_rel 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 RTC6, 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 119). • Jump commands initiating a jump to themselves (Pos = 0) 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 Stop). • Decisive are the runtime conditions. When reconfiguring list memory or converting a subroutine, an originally valid jump address might become invalid due to new list boundaries or a relocated subroutine storage position.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	list_jump_pos, list_jump_rel_cond

Undelayed Short List Command	list_jump_rel_cond
Function	<p><i>Conditional relative list jump:</i> list_jump_rel_cond executes list_jump_rel(<i>Pos</i>), if the current <i>IOvalue</i> 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>(= if the bits specified in <i>Mask1</i> are 1 and the bits specified in <i>Mask0</i> are 0). Otherwise, the directly following list command is immediately executed.</p>
Call	<code>list_jump_rel_cond(Mask1, Mask0, Pos)</code>
Parameters	<p>Mask1 16-bit mask. As an unsigned 32-bit value. Only the lower 16 bits are evaluated.</p> <p>Mask0 See <i>Mask1</i>.</p> <p>Pos Jump distance $[(-2^{23}+1) \dots (2^{23}-1)]$. As a signed 32-bit value.</p>
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_rel_cond are allowed even if they are to their own address (<i>Pos</i> = 0), for example, to wait for confirmation of a signal. • See also Chapter 9.3.2 "Conditional Command Execution", page 294.
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 "Example Code (Pascal)", page 295.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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"> • <code>list_next</code> 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 $10 \mu\text{s}$ clock cycle. • <code>list_next</code> is a proper place holder for another command in the list, because it prevents an extra $10 \mu\text{s}$ clock cycle, which is unavoidable with other place holders such as <code>list_nop</code> or <code>list_continue</code>.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	list_nop , list_continue

Normal List Command	list_nop
Function	Inserts a null operation (no operation) into the list memory.
Call	<code>list_nop()</code>
Comments	<ul style="list-style-type: none"> • <code>list_nop</code> has the same effect as <code>long_delay(1)</code>. It switches off the Signals for "Laser Active" Operation after a LaserOff Delay and waits for a possible scanner delay. Even if no delays need to be waited for, execution of <code>list_nop</code> requires $10 \mu\text{s}$. • <code>list_nop</code> serves as a placeholder for rejected list commands (get_last_error return code <code>RTC6_IGNORED</code>). • When <code>list_nop</code> immediately follows a short list command, it ensures that the subsequent list command only executes in the next $10 \mu\text{s}$ clock cycle (the short list command's "effective" execution time is then $10 \mu\text{s}$).
RTC4→RTC6	The RTC4 command is a pure placeholder and is more like <code>list_continue</code> . The RTC6 command switches the Signals for "Laser Active" Operation off and waits for a scanner delay.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 121. Any still-pending delayed short list command executes first. All list commands between <code>list_repeat</code> and the next <code>list_until</code> call is possibly repeated multiple times. Each executed <code>list_repeat...list_until</code> loop requires a full 10 μs clock cycle. Accordingly, the first list command after a repetition is executed with a 10 μs delay. Nesting up to 8 <code>list_repeat...list_until</code> loops deep is allowed. Each further <code>list_repeat</code> call beyond that limit is ignored as long as no <code>list_until</code> call has terminated a loop. If a list terminates (by <code>set_end_of_list</code> or <code>stop_execution</code> or /STOP), then all not-yet-fully-processed <code>list_repeat...list_until</code> loops are deleted. However, this does not occur if an automatic list change (by <code>auto_change</code>, <code>auto_change_pos</code> or <code>start_loop</code>) is active. Complete <code>list_repeat...list_until</code> 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 <code>list_jump_pos</code>). List jumps into a <code>list_repeat...list_until</code> loop 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→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	list_until

Undelayed Short List Command	list_return
Function	Terminates a previously called subroutine, jumps to the calling location and executes the immediately following list command (possibly after a list_continue) 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 list memory area "List 3", 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 list memory area "List 3", 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 list_return, a flush of the buffered list input is triggered, see Chapter 6.4.1 "Loading Lists", page 104. 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→RTC6	No changes to the previous functionality (termination of a subroutine). New: impact during loading into the protected list memory area "List 3".
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	list_call , sub_call , load_char , load_text_table , load_sub , list_continue



Undelayed Short List Command	list_until
Function	Terminates repetition of a group of list commands.
Call	<code>list_until(Number)</code>
Parameters	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 121. • 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 has been 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→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	list_repeat

Ctrl Command	load_auto_laser_control																				
Function	Loads a table with data points from an ASCII text file. Determines – by linear interpolation – the nonlinearity curve (see Section "Loading and Determining the Nonlinearity Curve", page 203) for "Automatic Laser Control" (except Vector-Defined Laser Control), see Chapter 7.4.9 ""Automatic Laser Control"" , page 196.																				
Call	NoOfDataPoints = load_auto_laser_control(Name, No)																				
Parameters	<table> <tr> <td>Name</td><td>Name of the text file or NULL. The text file may contain one or more tables. As a pointer to a \0-terminated ANSI string.</td></tr> <tr> <td>No</td><td>Determines which table in the text file is to be loaded. The parameter corresponds to the extension <No> of [AutoLaserCtrlTable<No>] at the beginning of the desired table. As an unsigned 32-bit value.</td></tr> </table>	Name	Name of the text file or NULL . The text file may contain one or more tables. As a pointer to a \0-terminated ANSI string.	No	Determines which table in the text file is to be loaded. The parameter corresponds to the extension <No> of [AutoLaserCtrlTable<No>] at the beginning of the desired table. As an unsigned 32-bit value.																
Name	Name of the text file or NULL . The text file may contain one or more tables. As a pointer to a \0-terminated ANSI string.																				
No	Determines which table in the text file is to be loaded. The parameter corresponds to the extension <No> of [AutoLaserCtrlTable<No>] at the beginning of the desired table. As an unsigned 32-bit value.																				
Result	<p>A positive error code in case of an error. The negative number of found data points in case of success. As a signed 32-bit value.</p> <table> <tr> <td>Value</td><td>Description</td></tr> <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 "Loading and Determining the Nonlinearity Curve", page 203.</td></tr> <tr> <td>-1024</td><td>For Name = NULL (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 list execution status error, board has been BUSY list execution status or INTERNAL-BUSY list execution status, no download (get_last_error return code RTC6_BUSY).</td></tr> <tr> <td>8</td><td>Board is locked by another user program (get_last_error return code RTC6_ACCESS_DENIED).</td></tr> <tr> <td>11</td><td>PCI error (get_last_error return code RTC6_SEND_ERROR), verify error (get_last_error return code RTC6_VERIFY_ERROR).</td></tr> <tr> <td>13</td><td>The specified table number could not be found in the file.</td></tr> </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 "Loading and Determining the Nonlinearity Curve", page 203 .	-1024	For Name = NULL (see also comments).	1	No valid data points found (though Table No found).	3	File not found.	4	DSP memory error.	5	BUSY list execution status error, board has been BUSY list execution status or INTERNAL-BUSY list execution status , no download (get_last_error return code RTC6_BUSY).	8	Board is locked by another user program (get_last_error return code RTC6_ACCESS_DENIED).	11	PCI error (get_last_error return code RTC6_SEND_ERROR), verify error (get_last_error return code RTC6_VERIFY_ERROR).	13	The specified table number could not be 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 "Loading and Determining the Nonlinearity Curve", page 203 .																				
-1024	For Name = NULL (see also comments).																				
1	No valid data points found (though Table No found).																				
3	File not found.																				
4	DSP memory error.																				
5	BUSY list execution status error, board has been BUSY list execution status or INTERNAL-BUSY list execution status , no download (get_last_error return code RTC6_BUSY).																				
8	Board is locked by another user program (get_last_error return code RTC6_ACCESS_DENIED).																				
11	PCI error (get_last_error return code RTC6_SEND_ERROR), verify error (get_last_error return code RTC6_VERIFY_ERROR).																				
13	The specified table number could not be 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 "Loading and Determining the Nonlinearity Curve", page 203. When loading the table, the RTC6 determines suitable values for the entire range of percent values. load_auto_laser_control overwrites any previously loaded nonlinearity curve. For Name = NULL (as during initialization by load_program_file), the function Scale(Percent)=1.0 is loaded for the complete percent range (no nonlinearity). 																				



Ctrl Command	load_auto_laser_control
Comments (cont'd)	<ul style="list-style-type: none"> • load_auto_laser_control is not executed (get_last_error return code RTC6_BUSY), if: <ul style="list-style-type: none"> – the BUSY list execution status is set – the INTERNAL-BUSY list execution status is set • load_auto_laser_control is even executed, if: <ul style="list-style-type: none"> – a list has been paused by set_wait (PAUSED list execution status set) • During the runtime of load_auto_laser_control, External Starts are suppressed. • Before loading a table, load_auto_laser_control performs a DSP memory check that produces error code 4 in case of error. • The table can be saved by create_dat_file.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_auto_laser_control, create_dat_file

Ctrl Command	load_char
Function	Assigns a desired index to a character defined by subsequent list commands, and loads the character into the protected list memory area "List 3".
Call	<code>load_char(Char)</code>
Parameters	Char Index of the indexed character. As an unsigned 32-bit value. Allowed value 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 \times 256 + ASCII number of the character (character sets are numbered 0 to 3). If Char > 1023 then load_char is ignored (get_last_error return code RTC6_PARAM_ERROR). The addresses in the protected list memory area "List 3" 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 call. This is a prerequisite for actual storage of the commands, entry of the start address into the internal management table, and initiating a flush of the buffered list input, see Chapter 6.4.1 "Loading Lists", page 104. Otherwise (the input pointer is altered without a preceding list_return), the character definition with this index is not available. An indexed character definition is not stored if the protected list memory area "List 3" has not been 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 117.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	list_return , load_sub , load_text_table



Ctrl Command	load_correction_file
Function	Loads the specified correction file into RTC6 memory. Then calls automatically select_cor_table with the most recently used parameter values (or the default parameter values).
Call	<code>ErrorNo = load_correction_file(Name, No, Dim)</code>
Parameters	<p>Name Name of the correction file. As a pointer to a \0-terminated ANSI string.</p> <p>No Number of the correction table. Allowed values: [1...8]. As an unsigned 32-bit value. See also number_of_correction_tables.</p> <p>Dim Determines whether the correction file content is stored as a 2D correction table or (if possible) as a 3D correction table (see also comments). As an unsigned 32-bit value. = 2: 2D and 3D correction files are stored as a 2D correction table (downgrade, if necessary). = 3: If the Option "3D" is enabled, 2D and 3D correction files are stored as a 3D correction table (upgrade, if necessary). If the Option "3D" is not enabled, like <code>Dim = 2</code>. Others: <code>Dim</code> 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 Option "3D" is enabled, otherwise as a 2D correction table (downgrade).</p>
Result	<p>Error code. As an unsigned 32-bit value.</p> <p>0 Success.</p> <p>1 File error (file corrupt or incomplete).</p> <p>2 Memory error (RTC6 DLL-internal, Windows system memory).</p> <p>3 File-open error (empty string submitted for <code>Name</code> parameter, file not found, etc.).</p> <p>4 DSP memory error.</p> <p>5 PCI download error (RTC6 board driver error), Ethernet download error.</p> <p>8 RTC6 board driver not found (get_last_error return code <code>RTC6_ACCESS_DENIED</code>).</p> <p>10 Parameter error (incorrect <code>No</code>).</p>

Ctrl Command	load_correction_file	
Result (cont'd)	11	Access error: board reserved for another user program (get_last_error return code <code>RTC6_ACCESS_DENIED</code>) or version check has detected an error (OUT or RTC version not compatible to the current RTC6 DLL version, get_last_error return code <code>RTC6_ACCESS_DENIED RTC6_VERSION_MISMATCH</code>).
	12	Warning: 3D correction table or <code>Dim</code> = 3 selected, but the Option "3D" 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 list execution status or INTERNAL-BUSY list execution status (get_last_error return code <code>RTC6_BUSY</code>).
	14	PCI upload error (RTC6 board driver error, only with download verification)
	15	Verify error (only with download verification).
Comments	<p>On loading correction tables:</p> <ul style="list-style-type: none"> The RTC6 can store 8 different correction tables at the same time, for example, for use in a multiple scan head configuration. Before storing a correction file, <code>load_correction_file</code> performs a DSP memory check that produces error code 4 in case of error. If the parameter <code>No</code> is out of range, then no correction table is loaded (return value 10). Users can further restrict the permitted range by number_of_correction_tables. The name of the to-be-loaded correction file must be passed to <code>load_correction_file</code> as a pointer to a \0-terminated ANSI string. If <code>Name</code> is passed as a NULL pointer, the corresponding table is replaced by a 1-to-1 table (for <code>Dim</code> = 3 and enabled Option "3D" with a 1-to-1 3D correction table, otherwise with a 1-to-1 2D table). An empty string ("") for <code>Name</code> result in an error return code of 3. If the Option "3D" is not enabled (default), then 2D and 3D correction files are stored as 2D correction tables – regardless of the value specified for <code>Dim</code>. The 3D data sections of 3D correction files are then ignored. If, on the other hand, the Option "3D" is enabled, then both 2D and 3D correction tables can be loaded. <ul style="list-style-type: none"> For <code>Dim</code> = 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 Option "3D" has been enabled). For <code>Dim</code> = 3, if the Option "3D" 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 <code>load_z_table</code> or <code>load_z_table_no</code>, see Chapter 7.3.4 "3D Image Field", page 169. All other values for <code>Dim</code> do not change the type of the correction file. 	

Ctrl Command	load_correction_file
Comments (cont'd)	<p>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 RTC6 to the scan head (or to both scan head connectors). <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 (<code>HeadA = 1, HeadB = 0</code>) or the values most recently used (after <code>load_program_file</code>) when having called <code>select_cor_table</code>. <code>load_correction_file</code> does not return to the user program until the <code>select_cor_table</code>-induced jump to the corrected galvanometer scanner position has been completed. See also Notes, page 175. <p>Others:</p> <ul style="list-style-type: none"> RTC5/RTC6 correction tables contain parameters that with an RTC4 must be looked-up in the supplied <code>*_ReadMe.txt</code> file first, and then must be entered manually into the user program. With the RTC6 these parameters can be read from the currently loaded correction tables (by <code>get_table_para</code>) or from the assigned correction tables (by <code>get_head_para</code>). Thus, they are directly integrated into a user program. During the runtime of <code>load_correction_file</code>: <ul style="list-style-type: none"> No other control commands are executed <code>External Starts</code> are suppressed <code>load_correction_file</code> is not executed (<code>get_last_error</code> return code <code>RTC6_BUSY</code>), if: <ul style="list-style-type: none"> the <code>BUSY</code> list execution status is set the <code>INTERNAL-BUSY</code> list execution status is set <code>load_correction_file</code> is even executed, if: <ul style="list-style-type: none"> a list has been paused by <code>set_wait</code> (<code>PAUSED</code> list execution status set) If an <code>RTC6_VERSION_MISMATCH</code> error occurs (return value 11), a RTC6 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 RTC6 DLL version must be reloaded by <code>load_program_file</code> (after <code>RTC6_ACCESS_DENIED</code>, the single-board command <code>load_program_file</code> does not get access for the board). Only afterward (and after the board has been reacquired by <code>acquire_rtc</code> or <code>select_rtc</code>) <code>load_correction_file</code> can be normally executed again.

Ctrl Command	load_correction_file
RTC4→RTC6	<p>Except its basic function and the parameters <code>Name</code> and <code>No</code>, the functionality of <code>load_correction_file</code> has substantially changed:</p> <ul style="list-style-type: none"> • <code>load_correction_file</code> 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, <i>up to 8</i> 3D correction tables can be stored in RTC6 memory (in the 3D-enabled version). However, two 3D correction tables can <i>not</i> be simultaneously assigned to the scan head connector (see <code>select_cor_table</code>). • 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>. • <code>select_cor_table</code> is automatically called, see comments above and Section "Notes", page 175. • <code>load_correction_file</code> does not return to the user program until the correction motion has been completed (see comments above).
RTC5→RTC6	<p>Changed functionality.</p> <ul style="list-style-type: none"> • <code>load_correction_file</code> lets you load 8 correction tables simultaneously into the RTC6 memory. • Overlaps no longer exist between memory areas used elsewhere.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	—

Ctrl Command	load_disk
Function	Loads into the protected list 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)
Parameters	<p>Name File name or NULL. As a pointer to a \0-terminated ANSI string.</p> <p>Mode Determines how the loading procedure is executed. As an unsigned 32-bit value.</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 Name = NULL, only the internal management tables are initialized as with Mode = 0 (no loading occurs, no "empty" binary file must be provided). Otherwise, load_disk 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 pertaining list commands are thereby written to the protected list memory area "List 3" and the 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 Mode > 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.

Ctrl Command	load_disk
Comments (cont'd)	<ul style="list-style-type: none"> Together with <code>save_disk</code>, <code>load_disk</code> can be used, for example, to defragment the protected list memory area "List 3" and to subsequently protect subroutines, see Section "Subsequent Protection and Conversion of Non-Indexed Subroutines", page 115 and Section "Index Management and Defragmentation", page 114. 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 = 0</code> and subsequent files are loaded with <code>Mode > 0</code>, 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. If, during loading, the end of the protected list memory area "List 3" 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 <code>load_char</code>, <code>load_text_table</code> or <code>load_sub</code> commands) are not stored. Before each <code>load_disk</code>, be sure that the memory configuration provides a sufficiently large protected list memory area "List 3" above the list areas ("List 1" and "List 2"). <code>save_disk</code> 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 RTC6 DLL, not from the board – see also get_config_list and Chapter 6.7.1 "Notes on Board Acquisition by a User Program", page 127). 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. <code>load_disk</code> is not executed, if: <ul style="list-style-type: none"> "List 3" has no memory area assigned, that is, $Mem1 + Mem2 = 2^{23}$ (<code>get_last_error</code> return code <code>RTC6_PARAM_ERROR</code>) the BUSY list execution status is set (<code>get_last_error</code> return code <code>RTC6_BUSY</code>) the INTERNAL-BUSY list execution status is set (<code>get_last_error</code>-Returncode <code>RTC6_BUSY</code>) <code>load_disk</code> is even executed, if: <ul style="list-style-type: none"> a list has only been paused by <code>set_wait</code> (PAUSED list execution status set). But users are responsible for ensuring that no still-needed commands are overwritten, for example, if <code>set_wait</code> has been called from an indexed subroutine. During the runtime of <code>load_disk</code>: <ul style="list-style-type: none"> No other commands can be executed External Starts are suppressed <code>load_disk</code> checks the version info saved by <code>save_disk</code> and compares it with the current runtime version. Both must match. Otherwise, <code>load_disk</code> is not executed (<code>get_last_error</code> return code <code>RTC6_VERSION_MISMATCH</code>).



Ctrl Command	load_disk
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 615: version check.
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 positioning stages, see Section "2D Encoder Compensation for xy Positioning Stages", page 248 .				
Call	NoOfDataPoints = <code>load_fly_2d_table(Name, No)</code>				
Parameters	<table> <tr> <td>Name</td> <td>Name of the text file or NULL. The text file may contain one or more tables. As a pointer to a \0-terminated ANSI string.</td> </tr> <tr> <td>No</td> <td>Determines which table in the text file is to be loaded. No corresponds to the extension <No> of [Fly2DTable<No>] at the beginning of the desired table. As an unsigned 32-bit value.</td> </tr> </table>	Name	Name of the text file or NULL . The text file may contain one or more tables. As a pointer to a \0-terminated ANSI string.	No	Determines which table in the text file is to be loaded. No corresponds to the extension <No> of [Fly2DTable<No>] at the beginning of the desired table. As an unsigned 32-bit value.
Name	Name of the text file or NULL . The text file may contain one or more tables. As a pointer to a \0-terminated ANSI string.				
No	Determines which table in the text file is to be loaded. No corresponds to the extension <No> of [Fly2DTable<No>] at the beginning of the desired table. As an unsigned 32-bit value.				
Result	<p>A positive error code in case of an error. The negative number of found data points in case of success. As a signed 32-bit value.</p> <ul style="list-style-type: none"> < 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 Section "For the 2D compensation tables, the following rules apply:", page 249. 0 For Name = NULL (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 list execution status error, board is BUSY list execution status or INTERNAL-BUSY list execution status, no download (get_last_error return code RTC6_BUSY). 8 Board is locked by another user program (get_last_error return code RTC6_ACCESS_DENIED). 11 PCI error (get_last_error return code RTC6_SEND_ERROR), verify error (get_last_error return code RTC6_VERIFY_ERROR). 13 The specified table number could not be found in the file. 				
Comments	<ul style="list-style-type: none"> • No = <No> loads table number 1; No = <No> + 65.536 loads table number 2. • The text file's data format requirements for reference points of the 2D encoder compensation table are described in Section "For the 2D compensation tables, the following rules apply:", page 249. The largest of these reference points should not exceed the range $-524,288\dots+524,287$ (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. • load_fly_2d_table overwrites a previously loaded table for 2D encoder compensation. • If Name = NULL, then a 0-correction table for 2D encoder compensation is loaded and the table is marked as invalid (see init_fly_2d). 				

Ctrl Command	load_fly_2d_table
Comments (cont'd)	<ul style="list-style-type: none"> • <code>load_fly_2d_table</code> is not executed (<code>get_last_error</code> return code <code>RTC6_BUSY</code>), if: <ul style="list-style-type: none"> – the BUSY list execution status is set – the INTERNAL-BUSY list execution status is set • <code>load_fly_2d_table</code> is even executed, if: <ul style="list-style-type: none"> – a list has been paused by <code>set_wait</code> (PAUSED list execution status set) • During the runtime of <code>load_fly_2d_table</code>, External Starts are suppressed. • Before loading a table, <code>load_fly_2d_table</code> performs a DSP memory check that produces error code 4 in case of error.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality. Additional: 2 tables can now be loaded simultaneously.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	<code>set_fly_2d</code> , <code>init_fly_2d</code> , <code>get_fly_2d_offset</code>

Ctrl 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	<code>NoOfDataPoints = load_jump_table(Name, No, PosAck, MinDelay, MaxDelay, ListPos)</code>												
Parameters	<table> <tr> <td>Name</td> <td>See <code>load_jump_table_offset</code>.</td> </tr> <tr> <td>No</td> <td>See <code>load_jump_table_offset</code>.</td> </tr> <tr> <td>PosAck</td> <td>See <code>load_jump_table_offset</code>.</td> </tr> <tr> <td>MinDelay</td> <td>See <code>load_jump_table_offset</code>.</td> </tr> <tr> <td>MaxDelay</td> <td>See <code>load_jump_table_offset</code>.</td> </tr> <tr> <td>ListPos</td> <td>See <code>load_jump_table_offset</code>.</td> </tr> </table>	Name	See <code>load_jump_table_offset</code> .	No	See <code>load_jump_table_offset</code> .	PosAck	See <code>load_jump_table_offset</code> .	MinDelay	See <code>load_jump_table_offset</code> .	MaxDelay	See <code>load_jump_table_offset</code> .	ListPos	See <code>load_jump_table_offset</code> .
Name	See <code>load_jump_table_offset</code> .												
No	See <code>load_jump_table_offset</code> .												
PosAck	See <code>load_jump_table_offset</code> .												
MinDelay	See <code>load_jump_table_offset</code> .												
MaxDelay	See <code>load_jump_table_offset</code> .												
ListPos	See <code>load_jump_table_offset</code> .												
Result	See <code>load_jump_table_offset</code> .												
Comments	<ul style="list-style-type: none"> • <code>load_jump_table</code> is identical to <code>load_jump_table_offset</code> with <code>Offset = 0</code>. 												
RTC4→RTC6	New command.												
RTC5→RTC6	Unchanged functionality.												
Version info	Available as of DLL 600, OUT 600, RBF 600.												
Reference	<code>load_jump_table_offset</code>												

Ctrl 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(<i>Name</i> , <i>No</i> , <i>PosAck</i> , <i>Offset</i> , <i>MinDelay</i> , <i>MaxDelay</i> , <i>ListPos</i>)														
Parameters	<table> <tr> <td><i>Name</i></td><td>Name of the text file or NULL. The text file may contain one or more tables. As a pointer to a \0-terminated ANSI string.</td></tr> <tr> <td><i>No</i></td><td>Specifies: <ul style="list-style-type: none"> For <i>Name</i> = Filename, which table of the text file should be loaded. The parameter corresponds to the extension <No> of [JumpTable<No>] at the beginning of the desired table. For <i>Name</i> = NULL, which scan system (or which scan head connector) should be used for automatic determination. Allowed values: [1, 2]. As an unsigned 32-bit value. </td></tr> <tr> <td><i>PosAck</i></td><td>Position tolerance value in [bits] (only relevant for automatic determination). As an unsigned 32-bit value.</td></tr> <tr> <td><i>Offset</i></td><td>Offset in [10 μs] which is added to all automatically determined delay values (only relevant for automatic determination). As a signed 32-bit value.</td></tr> <tr> <td><i>MinDelay</i></td><td>Minimum Jump Delay in [10 μs] (only relevant for automatic determination). As an unsigned 32-bit value.</td></tr> <tr> <td><i>MaxDelay</i></td><td>Maximum Jump Delay in [10 μs] (only relevant for automatic determination). As an unsigned 32-bit value.</td></tr> <tr> <td><i>ListPos</i></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). As an unsigned 32-bit value.</td></tr> </table>	<i>Name</i>	Name of the text file or NULL . The text file may contain one or more tables. As a pointer to a \0-terminated ANSI string.	<i>No</i>	Specifies: <ul style="list-style-type: none"> For <i>Name</i> = Filename, which table of the text file should be loaded. The parameter corresponds to the extension <No> of [JumpTable<No>] at the beginning of the desired table. For <i>Name</i> = NULL, which scan system (or which scan head connector) should be used for automatic determination. Allowed values: [1, 2]. As an unsigned 32-bit value.	<i>PosAck</i>	Position tolerance value in [bits] (only relevant for automatic determination). As an unsigned 32-bit value.	<i>Offset</i>	Offset in [10 μ s] which is added to all automatically determined delay values (only relevant for automatic determination). As a signed 32-bit value.	<i>MinDelay</i>	Minimum Jump Delay in [10 μ s] (only relevant for automatic determination). As an unsigned 32-bit value.	<i>MaxDelay</i>	Maximum Jump Delay in [10 μ s] (only relevant for automatic determination). As an unsigned 32-bit value.	<i>ListPos</i>	List position (in the area of list 1 or 2) for six list commands that can be overwritten (only relevant for automatic determination). As an unsigned 32-bit value.
<i>Name</i>	Name of the text file or NULL . The text file may contain one or more tables. As a pointer to a \0-terminated ANSI string.														
<i>No</i>	Specifies: <ul style="list-style-type: none"> For <i>Name</i> = Filename, which table of the text file should be loaded. The parameter corresponds to the extension <No> of [JumpTable<No>] at the beginning of the desired table. For <i>Name</i> = NULL, which scan system (or which scan head connector) should be used for automatic determination. Allowed values: [1, 2]. As an unsigned 32-bit value.														
<i>PosAck</i>	Position tolerance value in [bits] (only relevant for automatic determination). As an unsigned 32-bit value.														
<i>Offset</i>	Offset in [10 μ s] which is added to all automatically determined delay values (only relevant for automatic determination). As a signed 32-bit value.														
<i>MinDelay</i>	Minimum Jump Delay in [10 μ s] (only relevant for automatic determination). As an unsigned 32-bit value.														
<i>MaxDelay</i>	Maximum Jump Delay in [10 μ s] (only relevant for automatic determination). As an unsigned 32-bit value.														
<i>ListPos</i>	List position (in the area of list 1 or 2) for six list commands that can be overwritten (only relevant for automatic determination). As an unsigned 32-bit value.														
Result	Error code. As a signed 32-bit value. <table> <tr> <td>-1...-50</td><td>Success for <i>Name</i> = 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 Section "Notes on Loading Determined Jump Delay Values", page 218.</td></tr> <tr> <td>-1024</td><td>Success for <i>Name</i> = NULL: Table has been automatically determined.</td></tr> <tr> <td>1</td><td>No valid data points found (but Table <i>No</i> found).</td></tr> <tr> <td>3</td><td>File not found.</td></tr> <tr> <td>4</td><td>Verify error: DSP memory error.</td></tr> <tr> <td>5</td><td>Busy error, board is BUSY list execution status or INTERNAL-BUSY list execution status, no download (get_last_error return code RTC6_BUSY).</td></tr> <tr> <td>8</td><td>Access error: board reserved for another user program (get_last_error return code RTC6_ACCESS_DENIED).</td></tr> </table>	-1...-50	Success for <i>Name</i> = 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 Section "Notes on Loading Determined Jump Delay Values", page 218 .	-1024	Success for <i>Name</i> = NULL : Table has been automatically determined.	1	No valid data points found (but Table <i>No</i> found).	3	File not found.	4	Verify error: DSP memory error.	5	Busy error, board is BUSY list execution status or INTERNAL-BUSY list execution status , no download (get_last_error return code RTC6_BUSY).	8	Access error: board reserved for another user program (get_last_error return code RTC6_ACCESS_DENIED).
-1...-50	Success for <i>Name</i> = 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 Section "Notes on Loading Determined Jump Delay Values", page 218 .														
-1024	Success for <i>Name</i> = NULL : Table has been automatically determined.														
1	No valid data points found (but Table <i>No</i> found).														
3	File not found.														
4	Verify error: DSP memory error.														
5	Busy error, board is BUSY list execution status or INTERNAL-BUSY list execution status , no download (get_last_error return code RTC6_BUSY).														
8	Access error: board reserved for another user program (get_last_error return code RTC6_ACCESS_DENIED).														

Ctrl Command	load_jump_table_offset
Result (cont'd)	10 Only if <code>Name</code> = NULL : Param error: HeadNo or ListPos invalid. 11 PCI error during download (<code>get_last_error</code> return code RTC6_SEND_ERROR). 13 The specified table number could not be found in the file.
Comments	<ul style="list-style-type: none"> For information on command usage, see Section "Jump-Length-Dependent Jump Delays", page 217. See also Chapter 8.1.5 "Jump Mode", page 216. Format requirements for placing the table with Jump Delay data points into the text file are described in Section "Notes on Loading Determined Jump Delay Values", page 218. When loading the table, the RTC6 uses linear interpolation to establish appropriate values for the complete jump length range. load_jump_table_offset overwrites any previously loaded Jump Delay tables for Jump Mode. If <code>Name</code> = filename, then the table number <code>No</code> is loaded. The other parameters are not used. If <code>Name</code> = NULL, then the Jump Delay table for head <code>No</code> is automatically determined and loaded (if Jump Mode has been previously enabled and activated successfully by <code>set_jump_mode</code> (Flag = 1)). Here, the parameters <code>PosAck</code>, <code>Offset</code>, <code>MinDelay</code>, <code>MaxDelay</code> and <code>ListPos</code> are used (see Section "Automatic Determination of the Jump Delay Table", page 220). Jump Mode takes effect upon the next 2D jump only if it has been activated and enabled by <code>set_jump_mode</code> or activated by <code>set_jump_mode_list</code>. load_jump_table_offset is not executed (<code>get_last_error</code> return code RTC6_BUSY), if: <ul style="list-style-type: none"> the BUSY list execution status is set the INTERNAL-BUSY list execution status is set load_jump_table_offset is even executed, if: <ul style="list-style-type: none"> a list has been paused by <code>set_wait</code> (PAUSED list execution status set) During the runtime of load_jump_table_offset, 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. The table can be saved by <code>create_dat_file</code>.
RTC4→RTC6	New command. There is no RTC4 Compatibility Mode for this command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
Reference	load_jump_table , set_jump_mode , set_jump_mode_list , get_jump_table , set_jump_table , create_dat_file

Ctrl Command	load_list
Function	Opens the list memory for writing with list commands and sets the input pointer to the specified <i>relative</i> position within the desired list ("List 1" or "List 2"), but only if the list is not currently being processed (BUSY list status is not set) or has already been processed (USED list status is set).
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"> = 0: The list is opened for which the BUSY list status is not currently set. "List 1" is opened, if the BUSY list status is not set for either list. = 1: "List 1" is opened, if the BUSY list status for it is not set (BUSY1 not set). = 2: "List 2" is opened if the BUSY list status for it is not set (BUSY2 not set). = 3: The list is opened for which the BUSY list status is not currently set but the USED list status. If for both lists the BUSY list status is not set but the USED list status: 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:</p> <ul style="list-style-type: none"> • the BUSY list status for it is not set (ListNo = 0...2) • the BUSY list status for it is not set but the USED list status (ListNo = 3) <p>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...(2²³-1)].</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"> • If load_list has been successful, the next list command is stored at the specified address and all subsequent list commands at the following addresses in the selected list. • If load_list has not been 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 (for example, by repeating load_list with a positive result or by the set_start_list_pos command etc.). It is the responsibility of the user to react to a return code of 0. load_list produces <i>no</i> wait loop and <i>does not</i> block execution of subsequent control commands.



Ctrl Command	load_list
Comments (cont'd)	<ul style="list-style-type: none"> • load_list (ListNo = 3) is useful in scenarios such as alternating list changes, where you want to wait specifically for a list to be processed (see Section "Alternating List Changes", page 110) 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 list status not set and, if applicable, USED list status set), load_list behaves like set_start_list_pos (see comments there). • If ListNo = 0...2 when using load_list, then note that the BUSY list status of a list can change between opening of the list with load_list and the actual loading of list commands, for example, if in the meantime an automatic list change has occurred that has been 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 ListNo = 3 load_list ensures that a list is actually opened for loading only directly after processing (and after any successful automatic list changes). Particularly, if for no list the BUSY list status is set but for both the USED list status (for example, after initialization, after stop_execution or after an External Stop), the opened list number is synchronized with the following automatic list change.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 198).																				
Call	<code>NoOfDataPoints = load_position_control(Name, No)</code>																				
Parameters	<table> <tr> <td>Name</td> <td>Name of the text file or NULL. The text file may contain one or more tables. As a pointer to a \0-terminated ANSI string.</td> </tr> <tr> <td>No</td> <td>Determines which table in the text file is to be loaded. The parameter corresponds to the extension <code><No></code> of [PositionCtrlTable<No>] at the beginning of the desired table. As an unsigned 32-bit value.</td> </tr> </table>	Name	Name of the text file or NULL . The text file may contain one or more tables. As a pointer to a \0-terminated ANSI string.	No	Determines which table in the text file is to be loaded. The parameter corresponds to the extension <code><No></code> of [PositionCtrlTable<No>] at the beginning of the desired table. As an unsigned 32-bit value.																
Name	Name of the text file or NULL . The text file may contain one or more tables. As a pointer to a \0-terminated ANSI string.																				
No	Determines which table in the text file is to be loaded. The parameter corresponds to the extension <code><No></code> of [PositionCtrlTable<No>] at the beginning of the desired table. As an unsigned 32-bit value.																				
Result	<p>A positive error code in case of an error. The negative number of found data points in case of success. As a signed 32-bit value.</p> <table> <tr> <td>Value</td> <td>Description</td> </tr> <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 198.</td> </tr> <tr> <td>-256</td> <td>For <code>Name = NULL</code> (see also comments).</td> </tr> <tr> <td>1</td> <td>No valid data points found (though Table <code>No</code> 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 list execution status error, board is BUSY list execution status or INTERNAL-BUSY list execution status, no download (get_last_error return code <code>RTC6_BUSY</code>).</td> </tr> <tr> <td>8</td> <td>Board is locked by another user program (get_last_error return code <code>RTC6_ACCESS_DENIED</code>).</td> </tr> <tr> <td>11</td> <td>PCI error (get_last_error return code <code>RTC6_SEND_ERROR</code>), verify error (get_last_error return code <code>RTC6_VERIFY_ERROR</code>).</td> </tr> <tr> <td>13</td> <td>The specified table number could not been found in the file.</td> </tr> </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 198 .	-256	For <code>Name = NULL</code> (see also comments).	1	No valid data points found (though Table <code>No</code> found).	3	File not found.	4	DSP memory error.	5	BUSY list execution status error, board is BUSY list execution status or INTERNAL-BUSY list execution status, no download (get_last_error return code <code>RTC6_BUSY</code>).	8	Board is locked by another user program (get_last_error return code <code>RTC6_ACCESS_DENIED</code>).	11	PCI error (get_last_error return code <code>RTC6_SEND_ERROR</code>), verify error (get_last_error return code <code>RTC6_VERIFY_ERROR</code>).	13	The specified table number could not been 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 198 .																				
-256	For <code>Name = NULL</code> (see also comments).																				
1	No valid data points found (though Table <code>No</code> found).																				
3	File not found.																				
4	DSP memory error.																				
5	BUSY list execution status error, board is BUSY list execution status or INTERNAL-BUSY list execution status, no download (get_last_error return code <code>RTC6_BUSY</code>).																				
8	Board is locked by another user program (get_last_error return code <code>RTC6_ACCESS_DENIED</code>).																				
11	PCI error (get_last_error return code <code>RTC6_SEND_ERROR</code>), verify error (get_last_error return code <code>RTC6_VERIFY_ERROR</code>).																				
13	The specified table number could not been 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 Section "Notes on Loading a Scaling Function", page 198. When loading the table, the RTC6 determines suitable values for the entire range of control values. load_position_control overwrites any previously loaded scaling function for position-dependent laser control. For <code>Name = NULL</code> (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 has been initialized by set_auto_laser_control. <p>Position-dependent laser control is deactivated by set_auto_laser_control (<code>Ctrl = 0</code>) or by loading $Scale(Position)=1.0$. See also Section "Position-Dependent Laser Control", page 198.</p>																				



Ctrl Command	load_position_control
Comments (cont'd)	<ul style="list-style-type: none"> • load_position_control is not executed (get_last_error return code RTC6_BUSY), if: <ul style="list-style-type: none"> – the BUSY list execution status is set – the INTERNAL-BUSY list execution status is set • load_position_control is even executed, if: <ul style="list-style-type: none"> – a list has been paused by set_wait (PAUSED list execution status set) • During the runtime of load_position_control, External Starts are suppressed. • Before loading a table, load_position_control performs a DSP memory check that produces error code 4 in case of error. • The table can be saved by create_dat_file.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_auto_laser_control , create_dat_file

Ctrl Command	load_program_file
Function	<ul style="list-style-type: none"> • Resets the RTC6 (Software reset) • Initializes the list memory • Performs a DSP memory check • Loads RTC6RBF.rbf • Loads RTC6OUT.out (with RTC6 Ethernet Boards, RTC6ETH.out is loaded instead) • Loads RTC6DAT.dat • Starts the DSP
Call	ErrorNo = load_program_file(Path)
Parameters	Path Path name of the directory, where RTC6OUT.out , RTC6RBF.rbf and RTC6DAT.dat are. As a pointer to a \0-terminated string.
Result	<p>Error code. As an unsigned 32-bit value.</p> <p>0 Success.</p> <p>1 Not used.</p> <p>2 The board is not running. If a renewed call does not bring success, then a Hardware reset is necessary.</p> <p>3 RTC6DAT.dat or RTC6RBF.rbf not found.</p> <p>4 Not used.</p> <p>5 Not enough Windows memory.</p> <p>6 Another user program has already acquired the board.</p> <p>7 Version error: RTC6 DLL version, RTC6RBF.rbf version and RTC6OUT.out/RTC6ETH.out version are not compatible with each other.</p> <p>8 RTC6 board driver not found.</p> <p>9 Loading of RTC6OUT.out or RTC6ETH.out failed or has incorrect format or other error.</p> <p>10 Not used.</p> <p>11 FPGA firmware error: loading of RTC6RBF.rbf failed.</p> <p>12 Error opening/reading RTC6DAT.dat.</p> <p>13 Not used.</p> <p>14 DSP memory error (external).</p> <p>15 Verify memory error.</p> <p>16 DSP memory error (internal).</p> <p>17 Ethernet error. See eth_get_last_error and eth_get_error.</p> <p>18 Only RTC6 Ethernet Board: NAND memory error.</p>

Ctrl Command	load_program_file
Comments	<ul style="list-style-type: none"> • If <code>Path = NULL</code>, then the path of the user program's current working directory is used. Caution: It is not always the folder from which the user program has been launched. The current working directory can change, for example, when a file from another folder is selected by the Windows Explorer (unless the "NoChangeDir" flag has been set when incorporating the Explorer window into the user program). • After each hardware reset (powerup), the first user program must begin by issuing a <code>load_program_file</code> command during initialization of the RTC6 board, see Initializing the Board, page 97. <code>load_program_file</code> should also be executed (for example, if another user program acquires the board – see acquire_rtc) when the board needs to be returned to the default state. • If multiple RTC6 boards are connected as master and slave, then <code>load_program_file</code> must have been called on all boards prior to initializing and operating the individual boards with further commands, see Chapter 6.6.3 "Master/Slave Operation", page 123. • After execution of <code>load_program_file</code>: <ul style="list-style-type: none"> – The laser focus is in the center of the <code>Image</code> field (0 0) – The laser control is <i>deactivated</i> – On the state of the various output ports, see Chapter 7.4.1 "Enabling, Activating and Switching Laser Control Signals", page 183 (LASERON, LASER1, LASER2), Chapter 9.1.1 "16-Bit Digital Output Port", page 281 (EXTENSION 1 socket connector), Chapter 9.1.2 "8-Bit Digital Output Port", page 282 (EXTENSION 2 socket connector), Chapter 9.1.3 "2 Bit Digital Output Port", page 282 (LASER Connector), Chapter 9.1.4 "12-Bit Analog Output Port 1 and 2", page 282 (LASER Connector). • <i>Caution! In general, sporadic load_program_file calls in your user program:</i> <ul style="list-style-type: none"> – <i>Contradict the safe switch-on sequence prescribed in Chapter 5.5 "Safe Start-up and Shutdown Sequences", page 90</i> – <i>Pose the risk of personal injury and/or property damage (cases have been reported where lasers with poor electric have emitted)</i> <i>If you absolutely cannot refrain from sporadic load_program_file calls in your user program, you must implement appropriate messages that warn users accordingly and prompt them to take actions that prevent these hazards.</i> • <code>load_program_file</code> does not load correction tables. Even 1-to-1 tables therefore need to be explicitly requested, see <code>load_correction_file</code>. Already-loaded correction tables remain loaded after <code>load_program_file</code>. • <code>load_correction_file</code> assigns a correction table by <code>select_cor_table(1, 0)</code> but does not execute a galvanometer scanner motion to the corrected output position. • <code>load_program_file</code> only returns to the calling user program, when <code>DSP</code> initialization has been completed.

Ctrl Command	load_program_file
Comments (cont'd)	<ul style="list-style-type: none"> • load_program_file automatically executes stop_execution, if: <ul style="list-style-type: none"> – the BUSY list execution status is set – the INTERNAL-BUSY list execution status is set • The files RTC6OUT.out, RTC6ETH.out, RTC6RBF.rbf and RTC6DAT.dat are included in the RTC6 software package. For easy identifying and archiving of different software versions, the files are also delivered zipped (the zip file names RTC6<...>_<Version>.zip include the version numbers). Copy or unzip the three files (of desired version) to the hard drive of your PC. • Assorted versions of the RTC6 DLL and the files RTC6OUT.out, RTC6ETH.out, RTC6RBF.rbf and RTC6DAT.dat cannot be arbitrarily combined with another (each zip file in the RTC6 software includes a text file with version information). load_program_file performs a version compatibility check. If there is a version error, then the loaded programs remain in RTC6 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 RTC6_ACCESS_DENIED RTC6_VERSION_MISMATCH). To then load a correct program version, load_program_file can be called. Hereby, temporary access rights are requested and released after the download (if the board has not been acquired by another user program; load_program_file does not perform an acquire_rtc).
RTC4→RTC6	<ul style="list-style-type: none"> • The command parameter specifies a directory name with RTC6 (in contrast a file name with the RTC4). • load_program_file loads three files, with fixed formats and names (RTC6OUT.out, RTC6RBF.rbf, RTC6DAT.dat) (see above). • After execution of the command, the laser control is <i>deactivated</i>.
RTC5→RTC6	<p>Changed functionality.</p> <ul style="list-style-type: none"> • Notes on migrating the source code of RTC5 user programs, etc.: load_program_file is downward compatible with the RTC5 and can continue to be used without modification. Though you do not need to change your user program, keep in mind the following: <ul style="list-style-type: none"> – There is no checking of the BUSY list execution status (hence error code 13 is never be outputted). – Data transfer by the McBSP interface is deactivated without warning. – A running list is terminated without warning, similarly to the control command stop_execution. Observe notes there about mirror positions etc. – Other used error codes are provided as listed in the table above.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	–

Ctrl Command	load_stretch_table																						
Function	Loads a table with data pairs from an ASCII text file for enhanced 3D correction, see Section "Enhanced 3D Correction", page 239 .																						
Call	<code>NoOfDataPairs = load_stretch_table(Name, No, TableNo)</code>																						
Parameters	<p>Name Name of the text file or NULL. The text file may contain one or more tables. As a pointer to a \0-terminated ANSI string.</p> <p>No As a signed 32-bit value. <ul style="list-style-type: none"> For $No \geq 0$, this parameter specifies which table in the text file is to be loaded. The parameter corresponds to the extension $<No>$ of [StretchTable<No>] at the beginning of the desired table. $No < 0$: Reserved. </p> <p>TableNo The already loaded 3D correction table to which the extended correction is assigned. As a signed 32-bit value. Allowed value range: 1...8. See also number_of_correction_tables.</p>																						
Result	<p>A positive error code in case of an error. The negative number of found data pairs in case of success. As a signed 32-bit value.</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 list execution status error, board is BUSY list execution status or INTERNAL-BUSY list execution status, no download (get_last_error return code RTC6_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 RTC6_SEND_ERROR).</td> </tr> <tr> <td>13</td> <td>The specified table number could not be found in the file.</td> </tr> <tr> <td>15</td> <td>Verify error (get_last_error return code RTC6_VERIFY_ERROR, only possible with active download verification, see set_verify).</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 list execution status error, board is BUSY list execution status or INTERNAL-BUSY list execution status , no download (get_last_error return code RTC6_BUSY).	6	Data error: data pairs missing.	11	PCI download error (get_last_error return code RTC6_SEND_ERROR).	13	The specified table number could not be found in the file.	15	Verify error (get_last_error return code RTC6_VERIFY_ERROR , only possible with active download verification, see set_verify).
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 list execution status error, board is BUSY list execution status or INTERNAL-BUSY list execution status , no download (get_last_error return code RTC6_BUSY).																						
6	Data error: data pairs missing.																						
11	PCI download error (get_last_error return code RTC6_SEND_ERROR).																						
13	The specified table number could not be found in the file.																						
15	Verify error (get_last_error return code RTC6_VERIFY_ERROR , only possible with active download verification, see set_verify).																						
Comments	<ul style="list-style-type: none"> For details about enhanced 3D correction, see Section "Enhanced 3D Correction", page 239. The enhanced 3D correction is also switched when select_cor_table is called. A successfully loaded table activates the new enhanced 3D correction. Here, load_stretch_table overwrites a previously loaded table of the same TableNo. If Name is not NULL, but no table has been successfully read, then load_stretch_table returns an error code, but otherwise has no effect (that is, any previous successfully downloaded table remains valid). 																						



Ctrl Command	load_stretch_table
Comments (cont'd)	<ul style="list-style-type: none"> • If <code>Name</code> is NULL, then <code>load_stretch_table</code> disables any enhanced 3D correction enabled by a previous <code>load_stretch_table</code>. • <code>load_stretch_table</code> is not executed (<code>get_last_error</code> return code RTC6_BUSY), if: <ul style="list-style-type: none"> – the BUSY list execution status is set – the INTERNAL-BUSY list execution status is set • <code>load_stretch_table</code> is even executed, if: <ul style="list-style-type: none"> – a list has been paused by <code>set_wait</code> (PAUSED list execution status set) • During execution of <code>load_stretch_table</code>, External Starts are suppressed. • Before loading a table, <code>load_stretch_table</code> performs a DSP memory check that produces error code 4 in case of error.
RTC4→RTC6	New command.
RTC5→RTC6	Changed functionality. <ul style="list-style-type: none"> • Additional <code>TableNo</code> parameter. This allows that each 3D correction table can be assigned its own extended 3D correction.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	select_cor_table

Ctrl Command	load_sub
Function	Assigns the specified index to a subroutine defined by subsequent list commands and loads the subroutine into the protected list memory area "List 3".
Call	<code>load_sub(Index)</code>
Parameters	<p>Index Index of the indexed subroutine. As an unsigned 32-bit value. Allowed value range: [0...1023].</p>
Comments	<ul style="list-style-type: none"> Up to 1024 indexed subroutines can be stored. If <code>Index > 1023</code> then <code>load_sub</code> is ignored (<code>get_last_error</code> return code <code>RTC6_PARAM_ERROR</code>). The address in the protected list memory area "List 3" where the subroutine should be stored is automatically determined and internally managed. Indexed subroutines must be terminated by a <code>list_return</code> call. This is a prerequisite for actual storage of the commands, entry of the start address into the internal management table, and initiating a flush of the buffered list input, see Chapter 6.4.1 "Loading Lists", page 104. Otherwise (the input pointer is altered without a preceding <code>list_return</code>), the subroutine with this index is not available. An indexed subroutine is not stored if the protected list memory area "List 3" has not been previously configured for a sufficient size beyond "List 1" and "List 2". If <code>list_return</code> is the next command after <code>load_sub</code>, then the corresponding subroutine is deleted from the internal management table. Indexed subroutines can be called by the <code>sub_call</code> command along with the corresponding index (see Section "General Information on Calling Subroutines", page 113). Observe all notes in Section "Indexed Subroutines", page 112.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	list_return , sub_call , load_char , load_text_table

Ctrl Command	load_text_table
Function	Assigns the specified index to a text string defined by subsequent list commands and loads the text string into the protected list memory area "List 3".
Call	<code>load_text_table(Index)</code>
Parameters	Index Index of the indexed text string. As an unsigned 32-bit value. Allowed value 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 Index > 41 then load_text_table is ignored (get_last_error return code RTC6_PARAM_ERROR). Even if digits are defined by mark_text instead of as individual characters with mark_char, the character set can still be subsequently switched for this purpose (see select_char_set). The addresses in the protected list memory area "List 3" where the text string definitions are stored are automatically determined and internally managed. Management is independent of that for indexed subroutines (see load_sub) and character definitions (see load_char). Indexed text string definitions must be terminated with a list_return call. This is a prerequisite for actual storage of the commands, entry of the start address into the internal management table, and initiating a flush of the buffered list input, see Chapter 6.4.1 "Loading Lists", page 104. Otherwise (the input pointer is altered without a preceding list_return), the text string with this index is not available. An indexed text string definition is not stored if the protected list memory area "List 3" has not been previously configured for a sufficient size beyond "List 1" and "List 2". If list_return is the next command after load_text_table, 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 117.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 user-defined “Variable Mark Delay”, see Section “User-defined “Variable Polygon Delays””, page 152 .																				
Call	NoOfDataPoints = load_varpolydelay(Name, No)																				
Parameters	<table> <tr> <td>Name</td> <td>Name of the text file or NULL. The text file may contain one or more tables. As a pointer to a \0-terminated ANSI string.</td> </tr> <tr> <td>No</td> <td>Table in the text file which is to be loaded. The parameter corresponds to the extension <code><No></code> of [VarPolyTable<No>] at the beginning of the desired table. As an unsigned 32-bit value.</td> </tr> </table>	Name	Name of the text file or NULL . The text file may contain one or more tables. As a pointer to a \0-terminated ANSI string.	No	Table in the text file which is to be loaded. The parameter corresponds to the extension <code><No></code> of [VarPolyTable<No>] at the beginning of the desired table. As an unsigned 32-bit value.																
Name	Name of the text file or NULL . The text file may contain one or more tables. As a pointer to a \0-terminated ANSI string.																				
No	Table in the text file which is to be loaded. The parameter corresponds to the extension <code><No></code> of [VarPolyTable<No>] at the beginning of the desired table. As an unsigned 32-bit value.																				
Result	<p>A positive error code in case of an error. The negative number of found data points in case of success. As a signed 32-bit value.</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 Section “User-defined “Variable Polygon Delays””, page 152.</td> </tr> <tr> <td>-1024</td> <td>For <code>Name = NULL</code>: the table initialized according to $1 - \cos(\phi)$ has been internally loaded (as with program start), see Figure 40.</td> </tr> <tr> <td>1</td> <td>No valid data points found (though Table <code>No</code> 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 list execution status error, board is BUSY list execution status or INTERNAL-BUSY list execution status, no download (get_last_error return code <code>RTC6_BUSY</code>).</td> </tr> <tr> <td>8</td> <td>The board is locked by another user program (get_last_error return code <code>RTC6_ACCESS_DENIED</code>).</td> </tr> <tr> <td>11</td> <td>PCI error (get_last_error return code <code>RTC6_SEND_ERROR</code>), verify error (get_last_error return code <code>RTC6_VERIFY_ERROR</code>).</td> </tr> <tr> <td>13</td> <td>The specified table number could not be 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 Section “User-defined “Variable Polygon Delays””, page 152 .	-1024	For <code>Name = NULL</code> : the table initialized according to $1 - \cos(\phi)$ has been internally loaded (as with program start), see Figure 40 .	1	No valid data points found (though Table <code>No</code> found).	3	File not found.	4	DSP memory error.	5	BUSY list execution status error, board is BUSY list execution status or INTERNAL-BUSY list execution status , no download (get_last_error return code <code>RTC6_BUSY</code>).	8	The board is locked by another user program (get_last_error return code <code>RTC6_ACCESS_DENIED</code>).	11	PCI error (get_last_error return code <code>RTC6_SEND_ERROR</code>), verify error (get_last_error return code <code>RTC6_VERIFY_ERROR</code>).	13	The specified table number could not be 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 Section “User-defined “Variable Polygon Delays””, page 152 .																				
-1024	For <code>Name = NULL</code> : the table initialized according to $1 - \cos(\phi)$ has been internally loaded (as with program start), see Figure 40 .																				
1	No valid data points found (though Table <code>No</code> found).																				
3	File not found.																				
4	DSP memory error.																				
5	BUSY list execution status error, board is BUSY list execution status or INTERNAL-BUSY list execution status , no download (get_last_error return code <code>RTC6_BUSY</code>).																				
8	The board is locked by another user program (get_last_error return code <code>RTC6_ACCESS_DENIED</code>).																				
11	PCI error (get_last_error return code <code>RTC6_SEND_ERROR</code>), verify error (get_last_error return code <code>RTC6_VERIFY_ERROR</code>).																				
13	The specified table number could not be 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 user-defined “Variable Mark Delay” are described in Section “User-defined “Variable Polygon Delays””, page 152. When loading the table, the RTC6 determines suitable values for the entire range of angles by linear interpolation. load_varpolydelay overwrites any previously loaded table for the “Variable Mark Delay”. For <code>Name = NULL</code> (as during initialization by load_program_file), the internal (default) table for the “Variable Mark Delay” ($1 - \cos(\phi)$, see Figure 40) is loaded. 																				



Ctrl Command	load_varpolydelay
Comments (cont'd)	<ul style="list-style-type: none"> • load_varpolydelay is not executed (get_last_error return code RTC6_BUSY), if: <ul style="list-style-type: none"> – the BUSY list execution status is set (a list is being processed or has been halted by pause_list) – the INTERNAL-BUSY list execution status is set • load_varpolydelay is even executed, if: <ul style="list-style-type: none"> – a list has been paused by set_wait (PAUSED list execution status set) • During the runtime of load_varpolydelay, External Starts are suppressed. • Before loading a table, load_varpolydelay performs a DSP memory check that produces error code 4 in case of error. • The table can be saved by create_dat_file.
RTC4→RTC6	<p>Basically unchanged functionality. However:</p> <ul style="list-style-type: none"> • To return to the internal standard Mark Delay table, a reset or renewed program loading by load_program_file is no longer necessary (see Name = NULL above). • The ASCII text file can have any filename extension.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	load_program_file, set_delay_mode, create_dat_file



Ctrl Command	load_z_table						
Function	Like load_z_table_20b . However, for a focus length value l in RTC4 compatibility range $[-32,768\dots+32,767]$.						
Restriction	Like load_z_table_no_20b .						
Call	<code>ErrorNo = load_z_table(A, B, C)</code>						
Parameters	<table> <tr> <td>A</td> <td>Like load_z_table_no.</td> </tr> <tr> <td>B</td> <td>Like load_z_table_no.</td> </tr> <tr> <td>C</td> <td>Like load_z_table_no.</td> </tr> </table>	A	Like load_z_table_no .	B	Like load_z_table_no .	C	Like load_z_table_no .
A	Like load_z_table_no .						
B	Like load_z_table_no .						
C	Like load_z_table_no .						
Result	Like load_z_table_no_20b .						
Comments	<ul style="list-style-type: none"> Like load_z_table_no_20b. <code>load_z_table(A, B, C)</code> is synonymous with <code>load_z_table_no(A, B, C, 0)</code>. 						
RTC4→RTC6	Unchanged functionality. In addition: changed value ranges and error codes. If the correct calibration factors are used (see Chapter “3D Commands”, page 237), then the same ABC coefficients can be used on the same 3-axis scan system with the RTC4 and RTC6 boards.						
RTC5→RTC6	Unchanged functionality.						
Version info	Available as of DLL 600, OUT 600, RBF 600.						
References	get_z_distance , read_abc_from_file , write_abc_to_file , load_z_table_no , select_cor_table , load_z_table_20b						

Ctrl Command	load_z_table_20b						
Function	Loads coefficients A , B and C into the currently assigned 3D correction table. For a focus length value l in the RTC6 20-bit range $[-524,288\dots+524,287]$.						
Restriction	Like load_z_table_no_20b .						
Call	<code>ErrorNo = load_z_table_20b(A, B, C)</code>						
Parameters	<table> <tr> <td>A</td> <td>Coefficient A of the parabolic function $z_{\text{out}} = A + Bl + Cl^2$ which is used for calculating the Z output values (focus length value l in the RTC6 20-bit range $[-524,288\dots+524,287]$). As a 64-bit IEEE floating point value. Allowed value range: $[-1,073,741,824.0\dots+1,073,741,824.0]$. Out-of-range values are clipped to the boundary values.</td> </tr> <tr> <td>B</td> <td>Like A (analogously). Allowed value range: $[-2,048.0\dots+2,048.0]$.</td> </tr> <tr> <td>C</td> <td>Like A (analogously). Allowed value range: $[-1.0\dots+1.0]$.</td> </tr> </table>	A	Coefficient A of the parabolic function $z_{\text{out}} = A + Bl + Cl^2$ which is used for calculating the Z output values (focus length value l in the RTC6 20-bit range $[-524,288\dots+524,287]$). As a 64-bit IEEE floating point value. Allowed value range: $[-1,073,741,824.0\dots+1,073,741,824.0]$. Out-of-range values are clipped to the boundary values.	B	Like A (analogously). Allowed value range: $[-2,048.0\dots+2,048.0]$.	C	Like A (analogously). Allowed value range: $[-1.0\dots+1.0]$.
A	Coefficient A of the parabolic function $z_{\text{out}} = A + Bl + Cl^2$ which is used for calculating the Z output values (focus length value l in the RTC6 20-bit range $[-524,288\dots+524,287]$). As a 64-bit IEEE floating point value. Allowed value range: $[-1,073,741,824.0\dots+1,073,741,824.0]$. Out-of-range values are clipped to the boundary values.						
B	Like A (analogously). Allowed value range: $[-2,048.0\dots+2,048.0]$.						
C	Like A (analogously). Allowed value range: $[-1.0\dots+1.0]$.						
Result	Like load_z_table_no_20b .						
Comments	<ul style="list-style-type: none"> • <code>load_z_table_20b</code> is not executed (<code>get_last_error</code> return code <code>RTC6_BUSY</code>), if: <ul style="list-style-type: none"> – the BUSY list execution status is set – the INTERNAL-BUSY list execution status is set • <code>load_z_table_20b</code> is even executed, if: <ul style="list-style-type: none"> – a list has been paused by <code>set_wait</code> (PAUSED list execution status set) • <code>load_z_table_20b</code> should always be used after <code>load_correction_file</code>, since <code>load_correction_file</code> sets the three coefficients to the default values of the loaded correction table. • The <code>ABC</code> values are lost by <code>select_cor_table</code>. • See also other comments at load_z_table_no. 						
RTC4→RTC6	New command.						
RTC5→RTC6	New command.						
Version info	Available as of DLL 631 , OUT 632 .						
References	get_z_distance , read_abc_from_file_20b , write_abc_to_file_20b , load_z_table_no_20b , select_cor_table , load_z_table_no_20b						

Ctrl Command	load_z_table_no								
Function	Like load_z_table_no_20b . However, for a focus length value l in RTC4 compatibility range $[-32,768\dots+32,767]$.								
Restriction	Like load_z_table_no_20b .								
Call	<code>ErrorNo = load_z_table_no(A, B, C, No)</code>								
Parameters	<table> <tr> <td>A</td> <td>Coefficient A of the parabolic function $z_{out} = A + Bl + Cl^2$ which is used for calculating the Z output values (focus length value l in the RTC4 compatibility range $[-32,768\dots+32,767]$). As a 64-bit IEEE floating point value. Allowed value range: $[-67.108.864.0\dots+67.108.864.0]$. Out-of-range values are clipped to the boundary values.</td> </tr> <tr> <td>B</td> <td>Like A (analogously). Allowed value range: $[-2,048.0\dots+2,048.0]$.</td> </tr> <tr> <td>C</td> <td>Like A (analogously). Allowed value range: $[-16.0\dots+16.0]$.</td> </tr> <tr> <td>No</td> <td>Number of the 3D correction table to which the three coefficients A, B, C are to be assigned. See also number_of_correction_tables.</td> </tr> </table>	A	Coefficient A of the parabolic function $z_{out} = A + Bl + Cl^2$ which is used for calculating the Z output values (focus length value l in the RTC4 compatibility range $[-32,768\dots+32,767]$). As a 64-bit IEEE floating point value. Allowed value range: $[-67.108.864.0\dots+67.108.864.0]$. Out-of-range values are clipped to the boundary values.	B	Like A (analogously). Allowed value range: $[-2,048.0\dots+2,048.0]$.	C	Like A (analogously). Allowed value range: $[-16.0\dots+16.0]$.	No	Number of the 3D correction table to which the three coefficients A, B, C are to be assigned. See also number_of_correction_tables .
A	Coefficient A of the parabolic function $z_{out} = A + Bl + Cl^2$ which is used for calculating the Z output values (focus length value l in the RTC4 compatibility range $[-32,768\dots+32,767]$). As a 64-bit IEEE floating point value. Allowed value range: $[-67.108.864.0\dots+67.108.864.0]$. Out-of-range values are clipped to the boundary values.								
B	Like A (analogously). Allowed value range: $[-2,048.0\dots+2,048.0]$.								
C	Like A (analogously). Allowed value range: $[-16.0\dots+16.0]$.								
No	Number of the 3D correction table to which the three coefficients A, B, C are to be assigned. See also number_of_correction_tables .								
Result	Like load_z_table_no_20b .								
Comments	<ul style="list-style-type: none"> Like load_z_table_no_20b. <code>load_z_table_no(A, B, C, 0)</code> is synonymous with <code>load_z_table(A, B, C)</code>. <code>load_z_table_no(A, B, C, No)</code> is synonymous with <code>load_z_table_no_20b(A x 16, B, C x 1/16, No)</code>. <p>The values are lost after a select_cor_table or select_cor_table_list.</p>								
RTC4→RTC6	New command.								
RTC5→RTC6	New command.								
Version info	Available as of DLL 611, OUT 611, RBF 616.								
References	load_z_table , get_z_distance , read_abc_from_file , write_abc_to_file , select_cor_table								

Ctrl Command	load_z_table_no_20b																										
Function	Loads coefficients A , B and C and then assigns them to the 3D correction table No. For a focus length value l in the RTC6 20-bit range $[-524,288\dots+524,287]$.																										
Restriction	If the Option "3D" has not been enabled or a 3D correction table has not been assigned (see select_cor_table), then load_z_table_no_20b returns the error code 12 or 13 and otherwise has no effect.																										
Call	<code>ErrorNo = load_z_table_no_20b(A, B, C, No)</code>																										
Parameters	<table> <tr> <td>A</td> <td>Coefficient A of the parabolic function $z_{\text{out}} = A + Bl + Cl^2$ which is used for calculating the Z output values (focus length value l in the RTC6 20-bit range $[-524,288\dots+524,287]$). As a 64-bit IEEE floating point value. Allowed value range: $[-1,073,741,824.0\dots+1,073,741,824.0]$. Out-of-range values are clipped to the boundary values.</td> </tr> <tr> <td>B</td> <td>Like A (analogously). Allowed value range: $[-2,048.0\dots+2,048.0]$.</td> </tr> <tr> <td>C</td> <td>Like A (analogously). Allowed value range: $[-1.0\dots+1.0]$.</td> </tr> <tr> <td>No</td> <td>Number of the 3D correction table to which the three coefficients A, B, C are to be assigned. See also number_of_correction_tables.</td> </tr> </table>	A	Coefficient A of the parabolic function $z_{\text{out}} = A + Bl + Cl^2$ which is used for calculating the Z output values (focus length value l in the RTC6 20-bit range $[-524,288\dots+524,287]$). As a 64-bit IEEE floating point value. Allowed value range: $[-1,073,741,824.0\dots+1,073,741,824.0]$. Out-of-range values are clipped to the boundary values.	B	Like A (analogously). Allowed value range: $[-2,048.0\dots+2,048.0]$.	C	Like A (analogously). Allowed value range: $[-1.0\dots+1.0]$.	No	Number of the 3D correction table to which the three coefficients A, B, C are to be assigned. See also number_of_correction_tables .																		
A	Coefficient A of the parabolic function $z_{\text{out}} = A + Bl + Cl^2$ which is used for calculating the Z output values (focus length value l in the RTC6 20-bit range $[-524,288\dots+524,287]$). As a 64-bit IEEE floating point value. Allowed value range: $[-1,073,741,824.0\dots+1,073,741,824.0]$. Out-of-range values are clipped to the boundary values.																										
B	Like A (analogously). Allowed value range: $[-2,048.0\dots+2,048.0]$.																										
C	Like A (analogously). Allowed value range: $[-1.0\dots+1.0]$.																										
No	Number of the 3D correction table to which the three coefficients A, B, C are to be assigned. See also number_of_correction_tables .																										
Result	<p>Error code. As an unsigned 32-bit value.</p> <p>Error bits with values 1...64 can also occur combined, but not in conjunction with error code 11...15, which only occur separately. Warnings 12 and 13 are only returned if no other errors exist.</p> <table> <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 list execution status or INTERNAL-BUSY list execution status error; for exact reason see get_last_error).</td> </tr> <tr> <td>11</td> <td>Access denied.</td> </tr> <tr> <td>12</td> <td>Option "3D" is not enabled.</td> </tr> <tr> <td>13</td> <td>No 3D correction table is currently assigned.</td> </tr> <tr> <td>14</td> <td>RTC6 board driver not found.</td> </tr> <tr> <td>15</td> <td>Invalid table number (> number_of_correction_tables).</td> </tr> </table>	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 list execution status or INTERNAL-BUSY list execution status error; for exact reason see get_last_error).	11	Access denied.	12	Option "3D" is not enabled.	13	No 3D correction table is currently assigned.	14	RTC6 board driver not found.	15	Invalid table number (> number_of_correction_tables).
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 list execution status or INTERNAL-BUSY list execution status error; for exact reason see get_last_error).																										
11	Access denied.																										
12	Option "3D" is not enabled.																										
13	No 3D correction table is currently assigned.																										
14	RTC6 board driver not found.																										
15	Invalid table number (> number_of_correction_tables).																										



Ctrl Command	load_z_table_no_20b
Comments	<ul style="list-style-type: none"> • load_z_table_no_20b is only needed for re-calibrating the z axis in a 3-axis scan system. For adjusting corresponding coefficients see Section "Checking the z axis Calibration", page 169. Both positive and negative coefficients can be specified. The coefficients should preferably be chosen so that all z output values $z_{out} = A + BI + CI^2$ lie within RTC6 20-bit range [-524,288...+524,287]. • Prior to the next list command that directly follows load_z_table_no_20b, 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). • By load_z_table_no_20b, the three coefficients A, B, C can be assigned to the 3D correction table No. When select_cor_table is executed, they are switched as well. • If No = currently assigned table number, the current values are overwritten as well and thus immediately makes them available, given no list is being processed. Otherwise, they are only saved and are only available after a select_cor_table or select_cor_table_list. • If no list is being processed, No = 0 merely overwrites the current values. • load_z_table_no_20b(A, B, C, No) is synonymous with load_z_table_no(A x 1/16, B x 16, C x 16, No). The values are lost after a select_cor_table or select_cor_table_list. <p>The following applies to B and C: The allowed value range decreases accordingly.</p>
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 631 , OUT 632 .
References	load_z_table_no_20b , load_z_table_20b , get_z_distance , read_abc_from_file_20b , write_abc_to_file_20b , select_cor_table

Ctrl Command	load_zoom_correction_file
Comments	<ul style="list-style-type: none"> • load_zoom_correction_file is described, for example, in the manual for the intelliWELD II FT.



Normal List Command	long_delay
Function	Pauses further processing of the list for the specified time.
Call	<code>long_delay(Delay)</code>
Parameters	<p>Delay Delay time. In bits. As an unsigned 32-bit value. 1 bit equals 10 μs. Allowed value range: $0 \leq \text{delay} \leq (2^{32}-1)$.</p>
Comments	<ul style="list-style-type: none"> • long_delay switches off the Signals for "Laser Active" Operation after a LaserOff Delay, waits for a possible scanner delay and pauses further processing of the list for the specified time. • long_delay should always be called after changing the lamp current of a YAG laser to obtain a constant laser power. • list_nop corresponds to <code>long_delay(1)</code>.
RTC4→RTC6	Unchanged functionality. In addition: increased value range.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_defocus_list

Normal List Command	mark_abs
Function	Moves the laser focus at mark speed along a 2D vector from the current position to the specified position (absolute coordinate values) within a 2D Image field .
Call	<code>mark_abs(X, Y)</code>
Parameters	<p>X Absolute x coordinate of the mark vector end point. In bits. As a signed 32-bit value. Allowed value range: [-268,435,456...+268,435,455]. Out-of-range values are clipped to the boundary values. The complete value range is only usable as a virtual Image field for example, for (enabled) Processing-on-the-fly applications.</p> <p>Y Like X (analogously).</p>
Comments	<ul style="list-style-type: none"> If the mark speed has not been previously explicitly set by set_mark_speed or set_mark_speed_ctrl, then the marking is executed at a predefined mark speed of 1,000 <i>bits/ms</i>. The Signals for "Laser Active" Operation are automatically turned on at the beginning of the marking (or remain on after a directly preceding [*]mark[*] Command or "Arc" command). The defined Scanner Delays and Laser Delays are thereby taken into account, see Chapter 7.2 "Delay Settings – Coordinating Scan Head Control and Laser Control", page 144. Note that other delays are executed in Sky Writing mode. Exception: zero-length [*]mark[*] Commands, see Section "Notes", page 148.
RTC4→RTC6	Unchanged functionality. In addition: increased value range. In RTC4 Compatibility Mode , the RTC6 multiplies the values specified for X and Y by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality. In addition: increased value range.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 mark speed along a 3D vector from the current position to the specified position (absolute coordinate values) within the 3D image field .
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see select_cor_table), then mark_abs_3d has the same effect as mark_abs . However, split-up into Microsteps is calculated like a 3D command and hence influences the effective mark speed in the xy plane.
Call	mark_abs_3d(X, Y, Z)
Parameters	<p>X Absolute x coordinate of the mark vector end point. In bits. As a signed 32-bit value. Allowed value range: [-268,435,456...+268,435,455]. Out-of-range values are clipped to the boundary values. The complete value range is only usable as a virtual Image field for example, for (enabled) Processing-on-the-fly applications.</p> <p>Y Like X (analogously).</p> <p>Z Like X (analogously), except Allowed value range: [-524,288...+524,287].</p>
Comments	<ul style="list-style-type: none"> Except for the additional motion in the third dimension, mark_abs_3d functions similarly to mark_abs (see comments there).
RTC4→RTC6	Unchanged functionality. In addition: increased value range. In RTC4 Compatibility Mode , the RTC6 multiplies the values specified for X, Y and Z by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality. In addition: increased value range. In RTC5 Compatibility Mode , the RTC6 multiplies the value specified for Z by 16. The allowed value range decreases accordingly.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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>
Parameters	<p>Char Index of the indexed character to be marked. As an unsigned 32-bit value. Allowed value range: [0...1023]). The following applies: Char = character set number × 256 + ASCII number of the character (character sets are numbered 0...3).</p>
Comments	<ul style="list-style-type: none"> • mark_char reads the indexed character's starting address from the internal management table based on the supplied index and then calls list_call (see also the comments there), which then starts the corresponding command list. • mark_char 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_continue might be executed, see Section "Normal, Short, Variable and Multiple List Commands", page 301. • 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, get_char_pointer returns the value “-1” (that is, $2^{32}-1$). • If $\text{Char} > 1023$, then mark_char is, already during loading, replaced by a list_nop (get_last_error return code RTC6_PARAM_ERROR). • Absolute Vector commands 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[*] Commands, “Arc” commands 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 117.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 commands and "Arc" commands receive an offset (based on the current coordinates at the time of the call).
Call	<code>mark_char_abs(Char)</code>
Parameters	Char Index of the indexed character to be marked. As an unsigned 32-bit value. Allowed value range: [0...1023]). The following applies: Char = character set number × 256 + ASCII number of the character (character sets are numbered 0...3).
Comments	<ul style="list-style-type: none"> • mark_char_abs reads the indexed character's starting address from the internal management table based on the supplied index and then calls 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→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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	<code>mark_date(Part, Mode)</code>
Parameters	<p>Part 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 (4 digits). = 6: Month as numerals (January = 1,...). = 7: Day-of-the-week as numerals (Sunday = 1,...). <p>As an unsigned 32-bit value. Allowed value range: [0...7].</p> <p>Mode Selection of the format.</p> <p>As an unsigned 32-bit value. Allowed value range: [0...3].</p> <p>a) Only affects Part = 2, 4, 6 or 7: The number of leading zeros in the day number.</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) Only affects Part = 0, 2 or 4...7: Desired character set for marking digits.</p> <p>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 RTC6 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 Chapter 7.5 "Marking Dates, Times and Serial Numbers", page 209). 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 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 Section "Defining Indexed Text Strings for Time, Date and Serial Number", page 118). Non-defined text strings or characters are ignored (that is, 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 Chapter 6.5.2 "Character Sets and Text Strings", page 117). 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 RTC6_PARAM_ERROR). Absolute Vector commands 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[*] Commands, "Arc" commands 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 00:00 o'clock. Leap years are represented in both date styles.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 commands and "Arc" commands receive an offset (based on the current coordinates at the time of the call).				
Call	mark_date_abs(Part, Mode)				
Parameters	<table> <tr> <td>Part</td><td>See mark_date.</td></tr> <tr> <td>Mode</td><td>See mark_date.</td></tr> </table>	Part	See mark_date .	Mode	See mark_date .
Part	See mark_date .				
Mode	See mark_date .				
Comments	<ul style="list-style-type: none"> • mark_date_abs has the same effect as 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→RTC6	New command.				
RTC5→RTC6	Unchanged functionality.				
Version info	Available as of DLL 600, OUT 600, RBF 600.				
References	mark_date				

Normal List Command	mark_ellipse_abs
Function	Moves the laser focus at mark speed along an elliptical arc around the specified midpoint (absolute coordinate values) within a 2D Image field .
Call	<code>mark_ellipse_abs(X, Y, Alpha)</code>
Parameters	<p>X Absolute x coordinate of the ellipse midpoint. In bits. As a signed 32-bit value. Allowed value range: [-268,435,456...+268,435,455]. Out-of-range values are clipped to the boundary values.</p> <p>Y Like X (analogously).</p> <p>Alpha Angle between elliptical half-axis a (defined by set_ellipse) and the x axis (the angle is referenced to the positive x direction, positive angle values correspond to counterclockwise angles). In degrees. As a 64-bit IEEE floating point value. 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. Before execution of mark_ellipse_abs, its shape must have been specified by set_ellipse. For descriptions of the individual parameters, see also Section "Ellipse Commands", page 137. 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 in Section "Ellipse Commands", page 137. See also all comments for arc_abs.
RTC4→RTC6	<p>New command.</p> <p>In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for X and Y by 16. The allowed value range decreases accordingly.</p>
RTC5→RTC6	Unchanged functionality. In addition: increased value range.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 mark speed along an elliptical arc around the specified midpoint (relative coordinate values) within a 2D Image field .
Call	<code>mark_ellipse_rel(dx, dy, Alpha)</code>
Parameters	<p><code>dx</code> Relative x coordinate of the ellipse midpoint. In bits. As a signed 32-bit value. Allowed value range: [-268,435,456...+268,435,455]. Out-of-range values are clipped to the boundary values.</p> <p><code>dy</code> Like <code>dx</code> (analogously).</p> <p><code>Alpha</code> Angle between elliptical half-axis <code>a</code> (defined by set_ellipse) and the x axis (see mark_ellipse_abs). In degrees. As a 64-bit IEEE floating point value.</p>
Comments	<ul style="list-style-type: none"> The coordinates for the ellipse midpoint are to be specified as relative coordinates with respect to the current position. Otherwise, mark_ellipse_rel is identical to mark_ellipse_abs (see comments there).
RTC4→RTC6	<p>New command.</p> <p>In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for <code>dx</code> and <code>dy</code> by 16. The allowed value range decreases accordingly.</p>
RTC5→RTC6	Unchanged functionality. In addition: increased value range.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	mark_ellipse_abs , set_ellipse



Normal List Command	mark_rel
Function	Moves the laser focus at mark speed along a 2D vector from the current position to the specified position (relative coordinate values) within a 2D Image field .
Call	<code>mark_rel(dx, dy)</code>
Parameters	<p><code>dx</code> Relative x coordinate of the mark vector end point. In bits. As a signed 32-bit value. Allowed value range: [-268,435,456...+268,435,455]. Out-of-range values are clipped to the boundary values. The complete value range is only usable as a virtual Image field for example, for (enabled) Processing-on-the-fly applications.</p> <p><code>dy</code> Like <code>dx</code> (analogously).</p>
Comments	<ul style="list-style-type: none"> The coordinates for the mark vector end point are to be supplied as relative coordinates with respect to the current position. Otherwise, mark_rel is identical to mark_abs (see comments there).
RTC4→RTC6	Unchanged functionality. In addition: increased value range. In RTC4 Compatibility Mode , the RTC6 multiplies the values specified for <code>dx</code> and <code>dy</code> by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality. In addition: increased value range.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	mark_abs , mark_rel_3d , timed_mark_rel

Normal List Command	mark_rel_3d
Function	Moves the laser focus at mark speed along a 3D vector from the current position to the specified position (relative coordinate values) in the 3D image field .
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see select_cor_table), then mark_rel_3d has the same effect as mark_rel . However, split-up into Microsteps is calculated like a 3D command and hence influences the effective mark speed in the xy plane.
Call	mark_rel_3d(dx, dy, dz)
Parameters	<p>dx Relative x coordinate of the mark vector end point. In bits. As a signed 32-bit value. Allowed value range: [-268,435,456...+268,435,455]. Out-of-range values are clipped to the boundary values. The complete value range is only usable as a virtual Image field for example, for (enabled) Processing-on-the-fly applications.</p> <p>dy Like dx (analogously).</p> <p>dz Like dx (analogously), except Allowed value range: [-524,288...+524,287].</p>
Comments	<ul style="list-style-type: none"> The coordinates for the mark vector end point are to be supplied as relative coordinates with respect to the current position. Otherwise, mark_rel_3d is identical to mark_abs_3d (see comments there).
RTC4→RTC6	Unchanged functionality. In addition: increased value range. In RTC4 Compatibility Mode , the RTC6 multiplies the values specified for dx , dy and dz by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality. In addition: increased value range. In RTC5 Compatibility Mode , the RTC6 multiplies the value specified for dz by 16. The allowed value range decreases accordingly.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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.</p> $\text{Mode} = 20 \times M_1 + 10 \times M_2 + M_3$ <p>Allowed value 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 value 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, for example, after an aborted list to determine if the current number has been 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 Starts and thus the maximum number of markings. Here, all markings of all serial-number-sets contribute jointly to the count.

Normal List Command	<code>mark_serial</code>
Comments (cont'd)	<ul style="list-style-type: none"> If <code>Digits</code> = 0, then a "markless" marking is executed. If <code>M₂</code> = 0, then the serial number is incremented by 1 (any increment size defined by <code>set_serial_step</code> or <code>set_serial_step_list</code> are not used in this case!). This can be useful, if a single serial number is to be omitted (repeat n times if necessary; n = increment), but can also be used to indirectly increase the serial number by an <i>additional</i> increment. For each to-be-marked serial number digit, the <code>mark_serial</code> command reads the starting address of the corresponding indexed text string (or – for <code>M₁</code> = 1 – of the corresponding indexed character) from the internal management table and then calls <code>list_call</code> (see also the comments there) an appropriate number of times, which starts the corresponding command lists. The command lists must contain marking instructions for digits 0...9 (see Section "Defining Indexed Text Strings for Time, Date and Serial Number", page 118). Non-defined text strings or characters are ignored (that is, not marked). The called indexed text strings can also contain calls to indexed characters (<code>mark_char</code> or <code>mark_char_abs</code>) and complete texts (<code>mark_text</code> or <code>mark_text_abs</code>). In the latter case, the character set can be switched if needed (before marking by <code>mark_serial</code>) with <code>select_char_set</code> (see also Chapter 6.5.2 "Character Sets and Text Strings", page 117). For invalid <code>Mode</code> values, the <code>mark_serial</code> is, already during loading, replaced by a <code>list_nop</code> (<code>get_last_error</code> return code <code>RTC6_PARAM_ERROR</code>). Absolute <code>Vector commands</code> and "<code>Arc</code>" <code>commands</code> execute absolutely after being called with <code>mark_serial</code>. If serial number markings need to execute at various locations within the <code>Image</code> <code>field</code>, then the corresponding indexed text strings (or characters) can only contain relative <code>[*]mark[*] Commands</code>, "<code>Arc</code>" <code>commands</code> or <code>Jump commands</code> or <code>mark_serial_abs</code> must be used instead.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	<code>set_serial</code> , <code>set_serial_step</code> , <code>set_serial_step_list</code> , <code>get_list_serial</code> , <code>set_max_counts</code> , <code>get_counts</code> , <code>load_text_table</code> , <code>set_text_table_pointer</code> , <code>mark_serial_abs</code>



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 commands 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	<table> <tr> <td>Mode</td><td>See mark_serial.</td></tr> <tr> <td>Digits</td><td>See mark_serial.</td></tr> </table>	Mode	See mark_serial .	Digits	See mark_serial .
Mode	See mark_serial .				
Digits	See mark_serial .				
Comments	<ul style="list-style-type: none"> • mark_serial_abs has the same effect as 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→RTC6	New command.				
RTC5→RTC6	Unchanged functionality.				
Version info	Available as of DLL 600, OUT 600, RBF 600.				
References	mark_serial				

Variable List Command	mark_text
Function	Marks a \0-terminated string.
Call	<code>mark_text(Text)</code>
Parameters	Text PC memory address of the first character (byte) of the to-be-marked text string. As a pointer to a \0-terminated string.
Comments	<ul style="list-style-type: none"> The to-be-marked text (character sequence, byte array, \0-terminated string) must be terminated with a \0 character (0 byte, NULL). The \0 character itself is not marked. When a mark_text 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 (make sure that no undesired memory overflow of the respective memory area occurs). 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 mark_text by select_char_set. For the default setting, character set 0 is used. If the 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, 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 RTC6_REJECTED) and the indexed subroutine is not stored. Absolute Vector commands and "Arc" commands execute absolutely after being called by 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[*] Commands, "Arc" commands or Jump commands or mark_text_abs must be used instead. mark_text 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→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	mark_text_abs



Variable List Command	mark_text_abs
Function	Marks a \0-terminated string. In the called indexed characters (see below), any absolute Vector commands and "Arc" commands receive an offset (based on the current coordinates at the time of the call).
Call	<code>mark_text_abs(Text)</code>
Parameters	Text PC memory address of the first character (byte) of the to-be-marked text string. As a pointer to a \0-terminated string.
Comments	<ul style="list-style-type: none"> During processing of the individual mark_text_abs, the corresponding mark_char_abs (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→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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	<code>mark_time(Part, Mode)</code>
Parameters	<p>Part 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>As an unsigned 32-bit value. Allowed value range: [0...4].</p> <p>Mode Format.</p> <p>As an unsigned 32-bit value. Allowed value 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. 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>
Comments	<ul style="list-style-type: none"> Before marking times (after every boot-up), the RTC6 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 Chapter 7.5 "Marking Dates, Times and Serial Numbers", page 209. The complete time can be marked by multiple calls of the mark_time command.



Normal List Command	mark_time
Comments (cont'd)	<ul style="list-style-type: none"> • mark_time reads (according to the stored time and according to the selected time part) the starting address of the corresponding indexed text string (or – for <code>Part = 0...3</code> and <code>Mode = 2</code> or <code>3</code> – of the corresponding indexed character) from the internal management table and then calls 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 <code>0...9</code> and for a.m./p.m., see Section "Defining Indexed Text Strings for Time, Date and Serial Number", page 118. Non-defined text strings or characters are ignored (that is, 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 Chapter 6.5.2 "Character Sets and Text Strings", page 117. • If <code>Part > 4</code> and/or <code>Mode > 3</code>, then mark_time is, already during loading, replaced by a list_nop (get_last_error return code <code>RTC6_PARAM_ERROR</code>). • Absolute Vector commands 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[*] Commands, "Arc" commands or Jump commands or mark_time_abs must be used instead.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 commands 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	Part See mark_time .
	Mode See mark_time .
Comments	<ul style="list-style-type: none"> • mark_time_abs has the same effect as 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→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	mark_time

Ctrl Command	master_slave_config
Function	Sets settings for the master-slave interface of an RTC6 board.
Call	<code>master_slave_config(Config)</code>
Parameters	<p>Config As an unsigned 32-bit value.</p> <p>Bit #0 Suppression of /Slave-START signals and /Slave-STOP signals.</p> <ul style="list-style-type: none"> = 0: /Slave-START and /Slave-STOP signals via the master-slave interface are received and processed. = 1: /Slave-START and /Slave-STOP signals received by this board via the master-slave interface are ignored on this board, but are forwarded to other boards of the master-slave interface. <p>Bit #1 /STOP in case of master-slave faults.</p> <ul style="list-style-type: none"> = 0: The board does not react explicitly to a fault in the master-slave connection. Any effects are undefined. = 1: Triggers a /STOP, if the connection to the master card is interrupted. This occurs, for example, if load_program_file is executed on a card of the already synchronized master-slave chain, the master-slave cable is removed or the signal of the master-slave interface is electromagnetically disturbed. <p>Bit #2 Forwarding /STOP in case of master-slave faults.</p> <ul style="list-style-type: none"> = 0: A /STOP is not forwarded to other boards. = 1: If an error of the master-slave connection is detected and a /STOP is triggered (see Bit #1), this /STOP is forwarded to all still connected cards of the master-slave chain (also upwards to a master card).
Comments	<ul style="list-style-type: none"> • For usage of master_slave_config, see Chapter 6.6.3 "Master/Slave Operation", page 123. • The master-slave interface of a board should normally only be configured after successful synchronization. • If Bit #0 is already set before sync_slaves, this board cannot be synchronized to a master board with sync_slaves. • In certain other malfunction scenarios, fault-free operation may no longer be possible. In this case, a /STOP is always performed on the affected board, even if Bit #1 = 0. • If the connection to a master-slave synchronized board is disturbed, this can also result in further boards in the chain being disturbed, and thus execute a /STOP, even if Bit #2 is not set.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 609, OUT 609, RBF 613.
References	sync_slaves



Ctrl Command	mcbsp_init				
Function	Defines the data delays for transmitting and receiving data by the McBSP interface , see also Section "McBSP Interface", page 83 .				
Call	<code>mcbsp_init(XDelay, RDelay)</code>				
Parameters	<table> <tr> <td>XDelay</td> <td>Transmission delay. As an unsigned 32-bit value. Allowed value range: [0...2].</td> </tr> <tr> <td>RDelay</td> <td>Receiving delay. As an unsigned 32-bit value. Allowed value range: [0...2].</td> </tr> </table>	XDelay	Transmission delay. As an unsigned 32-bit value. Allowed value range: [0...2].	RDelay	Receiving delay. As an unsigned 32-bit value. Allowed value range: [0...2].
XDelay	Transmission delay. As an unsigned 32-bit value. Allowed value range: [0...2].				
RDelay	Receiving delay. As an unsigned 32-bit value. Allowed value range: [0...2].				
Comments	<ul style="list-style-type: none"> For invalid parameter values, mcbsp_init is not executed (get_last_error return code RTC6_PARAM_ERROR). The initialized values (after program start) are XDelay = RDelay = 1. The signals and operating conditions of the McBSP interface are presented in Chapter 4.6.6 "McBSP/ANALOG Socket Connector", page 83. The McBSP interface ignores the first FrameSync signal after a mcbsp_init. That is, data provided is not transmitted, see page 85. 				
RTC4→RTC6	New command.				
RTC5→RTC6	Unchanged functionality.				
Version info	Available as of DLL 600, OUT 600, RBF 600.				
References	read_mcbsp , set_mcbsp_out , set_mcbsp_out_ptr , set_mcbsp_freq , mcbsp_init_spi				



Ctrl Command	mcbsp_init_spi
Function	No function.
Call	<code>mcbsp_init_spi(ClockLevel, ClockDelay)</code>
Parameters	<p><code>ClockLevel</code> = 0: inactive low. > 0: inactive high. As an 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. As an unsigned 32-bit value.</p>
Comments	<ul style="list-style-type: none"> See Chapter 4.6.6 "McBSP/ANALOG Socket Connector", page 83.
RTC4→RTC6	New command.
RTC5→RTC6	<p>Changed functionality.</p> <ul style="list-style-type: none"> The RTC6 command set contains <code>mcbsp_init_spi</code>, but it has no effect. Execution of it is refused and the <code>get_last_error</code> return code is <code>RTC6_REJECTED</code>. Notes on migrating the source code of RTC5 user programs, etc.: For hardware reasons, RTC6 boards are no longer configurable for <code>SPI</code> functionality. You need to appropriately modify any such source code sections.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 counter.				
Call	<code>measurement_status(&Busy, &Pos)</code>				
Returned parameter values	<table> <tr> <td>Busy</td> <td>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.</td> </tr> <tr> <td>Pos</td> <td>Current position of the measurement counter (within the RTC6 measurement data memory) [0...max. channel size, see set_trigger4]. As a pointer to an unsigned 32-bit value.</td> </tr> </table>	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 counter (within the RTC6 measurement data memory) [0...max. channel size, see set_trigger4]. As a pointer to an unsigned 32-bit value.
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 counter (within the RTC6 measurement data memory) [0...max. channel size, see set_trigger4]. As a pointer to an unsigned 32-bit value.				
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 ([0...max. channel size, see set_trigger4]). <code>Pos = 2³²-1</code> 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(Period = 0), set_trigger4(Period = 0), stop_trigger or stop_execution (see set_trigger comments). 				
RTC4→RTC6	Unchanged functionality.				
RTC5→RTC6	Unchanged functionality.				
Version info	Available as of DLL 600, OUT 600, RBF 600.				
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 split-up into Microsteps) directly from the current position to the specified position (absolute coordinate values) within the 2D Image field .
Call	micro_vector_abs(X , Y , LasOn , LasOff)
Parameters	<p>X Absolute x coordinate of the micro vector end point. In bits. As a signed 32-bit value. Allowed value range: [-268,435,456...+268,435,455]. Out-of-range values are clipped to the boundary values. The complete value range is only usable as a virtual Image field for example, for (enabled) Processing-on-the-fly applications.</p> <p>Y Like X (analogously).</p> <p>LasOn LaserOn Delay. 1 bit equals 1/64 μs. As a signed 32-bit value. Allowed value range: [-1...+(2¹⁵-1)]. ≥ 0: Delay is newly set. Values over (2¹⁵-1) are clipped. < 0: The previously set delay continues unaffected.</p> <p>LasOff LaserOff Delay. 1 bit equals 1/64 μs. See LasOn.</p>
Comments	<ul style="list-style-type: none"> See also Chapter 8.8 “micro_vector[*] Commands”, page 272. Wobbel is not taken into account, see 2 in Chapter 7.3.6 “Output Values to the Scan System”, page 180. The Microvector is always executed as a “Hard jump”. By LasOn ≥ 0 and LasOff ≥ 0, you can set a new LaserOn Delay or LaserOff Delay for each individual Microvector. Each delay thereby gets set at the end of the clock cycle in which the new position actually gets outputted (this output clock cycle is delayed by a preceding scanner delay). Negative values (LasOn < 0 and LasOff < 0) do not affect Laser Delays. Hereby, the laser can remain on or off across multiple clock cycles (mark and jump simulation). Delays set with LasOn and LasOff only apply to the execution of Microvectors. For execution of normal [*]mark[*] Commands and “Arc” commands (such as mark_abs), only the Laser Delays defined by set_laser_delays apply. LasOn and LasOff do not overwrite the laser delay parameter from set_laser_delays.
RTC4→RTC6	<p>New command.</p> <p>In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for X and Y by 16, those for LasOn and LasOff by 64. The allowed value ranges decrease accordingly.</p>
RTC5→RTC6	<p>Unchanged functionality.</p> <p>In RTC5 Compatibility Mode, the RTC6 multiplies the specified values for LasOn and LasOff by 32. The allowed value ranges decrease accordingly.</p>
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 split-up into Microsteps) directly from the current position to the specified position (absolute coordinate values) within the 3D image field .
Restriction	If the Option “3D” is not enabled or no 3D correction table has been assigned (see select_cor_table), then micro_vector_abs_3d has the same effect as micro_vector_abs .
Call	<code>micro_vector_abs_3d(X, Y, Z, LasOn, LasOff)</code>
Parameters	<p>X Absolute x coordinate of the micro vector end point. In bits. As a signed 32-bit value. Allowed value range: [-268,435,456...+268,435,455]. Out-of-range values are clipped to the boundary values. The complete value range is only usable as a virtual Image field for example, for (enabled) Processing-on-the-fly applications.</p> <p>Y Like X (analogously).</p> <p>Z Like X (analogously), except Allowed value range: [-524,288...+524,287].</p> <p>LasOn LaserOn Delay. 1 bit equals 1/64 μs. As a signed 32-bit value. Allowed value range: [-1...+(2¹⁵-1)]. ≥ 0: Delay is newly set. Values over (2¹⁵-1) are clipped. < 0: The previously set delay continues unaffected.</p> <p>LasOff LaserOff Delay. Like LasOn.</p>
Comments	<ul style="list-style-type: none"> Except for the additional motion in the third dimension, micro_vector_abs_3d functions similarly to micro_vector_abs (see comments there).
RTC4→RTC6	New command. In RTC4 Compatibility Mode , the RTC6 multiplies the specified values for X , Y and Z by 16, those for LasOn and LasOff by 64. The allowed value ranges decrease accordingly.
RTC5→RTC6	Unchanged functionality. In RTC5 Compatibility Mode , the RTC6 multiplies the specified value for Z by 16, those for LasOn and LasOff by 32. The allowed value ranges decrease accordingly.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 split-up into Microsteps) directly from the current position to the specified position (relative coordinate values) within the 2D Image field .
Call	<code>micro_vector_rel(dX, dY, LasOn, LasOff)</code>
Parameters	<p>dX Relative x coordinate of the micro vector end point. In bits. Otherwise, like x from micro_vector_abs.</p> <p>dY Relative y coordinate of the micro vector end point. In bits. Otherwise, like y from micro_vector_abs.</p> <p>LasOn Like LasOn from micro_vector_abs.</p> <p>LasOff Like LasOff from micro_vector_abs.</p>
Comments	<ul style="list-style-type: none"> The coordinates for the micro vector end point are to be supplied as relative coordinates with respect to the current position. Otherwise, micro_vector_rel is identical to micro_vector_abs (see comments there). The Microvector is always executed as a “Hard jump”.
RTC4→RTC6	<p>New command.</p> <p>RTC4 Compatibility Mode: see micro_vector_abs.</p>
RTC5→RTC6	<p>Unchanged functionality.</p> <p>RTC5 Compatibility Mode: see micro_vector_abs.</p>
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 split-up into Microsteps) directly from the current position to the specified position (relative coordinate values) within the 3D image field .										
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see select_cor_table), then micro_vector_rel_3d has the same effect as micro_vector_rel .										
Call	<code>micro_vector_rel_3d(dx, dy, dz, LasOn, LasOff)</code>										
Parameters	<table> <tr> <td>dx</td><td>Relative x coordinate of the micro vector end point. In bits. Otherwise, like wie x von micro_vector_abs_3d.</td></tr> <tr> <td>dy</td><td>Relative y coordinate of the micro vector end point. In bits. Otherwise, like wie y von micro_vector_abs_3d.</td></tr> <tr> <td>dz</td><td>Relative z coordinate of the micro vector end point. In bits. Otherwise, like wie z von micro_vector_abs_3d.</td></tr> <tr> <td>LasOn</td><td>Like LasOn from micro_vector_abs_3d.</td></tr> <tr> <td>LasOff</td><td>Like LasOff from micro_vector_abs_3d.</td></tr> </table>	dx	Relative x coordinate of the micro vector end point. In bits. Otherwise, like wie x von micro_vector_abs_3d .	dy	Relative y coordinate of the micro vector end point. In bits. Otherwise, like wie y von micro_vector_abs_3d .	dz	Relative z coordinate of the micro vector end point. In bits. Otherwise, like wie z von micro_vector_abs_3d .	LasOn	Like LasOn from micro_vector_abs_3d .	LasOff	Like LasOff from micro_vector_abs_3d .
dx	Relative x coordinate of the micro vector end point. In bits. Otherwise, like wie x von micro_vector_abs_3d .										
dy	Relative y coordinate of the micro vector end point. In bits. Otherwise, like wie y von micro_vector_abs_3d .										
dz	Relative z coordinate of the micro vector end point. In bits. Otherwise, like wie z von micro_vector_abs_3d .										
LasOn	Like LasOn from micro_vector_abs_3d .										
LasOff	Like LasOff from micro_vector_abs_3d .										
Comments	<ul style="list-style-type: none"> The coordinates for the micro vector end point are to be supplied as relative coordinates with respect to the current position. Otherwise, micro_vector_rel_3d is identical to micro_vector_abs_3d (see comments there). The Microvector is always executed as a "Hard jump". 										
RTC4→RTC6	New command. RTC4 Compatibility Mode: see micro_vector_abs_3d .										
RTC5→RTC6	Unchanged functionality. RTC5 Compatibility Mode: see micro_vector_abs_3d .										
Version info	Available as of DLL 600, OUT 600, RBF 600.										
References	micro_vector_abs_3d , micro_vector_rel										

Ctrl Command	move_to
Comments	<ul style="list-style-type: none"> 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). move_to has been introduced for the "RTC4 STEP MOTOR EXTENSION" extension board (#112097). move_to is not supported by "RTC5/6 varioSCAN 40 FLEX Extension" extension board (#128683). See also Chapter 2.8.7 "Extension Board "RTC5/6 varioSCAN FLEX Extension"", page 45.



Ctrl Command	number_of_correction_tables
Function	Defines the maximum number of allowed correction tables.
Call	<code>number_of_correction_tables(Number)</code>
Parameters	<p>Number Maximum number of allowed correction tables. As an unsigned 32-bit value. Allowed value range: [1...8]. Default after load_program_file: 8. See also Chapter 8.5.3 "Using Several Correction Tables", page 240.</p>
Comments	<ul style="list-style-type: none"> For outside the allowed value range, number_of_correction_tables is ignored (get_last_error return code <code>RTC6_PARAM_ERROR</code>). number_of_correction_tables serves to protect other commands (for example, such as load_correction_file and select_cor_table) from unwanted table numbers. Existing user programs do not have to be changed. The exception is, if user input is to be rejected (using explicit RTC6 error messages) in the future. number_of_correction_tables refers only to subsequent command executions. Existing assignments of correction tables cannot be corrected automatically. This is particularly important, if RTC6 boards that have been initialized by other user programs are subsequently acquired.
RTC4→RTC6	New command.
RTC5→RTC6	Changed impacts (the RTC5-command has not been documented due to the restricted user group).
Version info	Available as of DLL 609, OUT 609, RBF 613.
References	load_correction_file , select_cor_table , select_cor_table_list



Normal List Command	para_jump_abs
Function	Moves the output point for the laser focus along a 2D vector at jump speed from the current position to the specified position (absolute coordinate values) within a 2D Image field . Simultaneously varies the signal parameter selected by set_vector_control linearly to the specified value.
Call	para_jump_abs(X , Y , P)
Parameters	<p>X Absolute x coordinate of the jump vector end point. In bits. As a signed 32-bit value. Allowed value range: [-268,435,456...+268,435,455]. Out-of-range values are clipped to the boundary values. The complete value range is only usable as a virtual Image field for example, for (enabled) Processing-on-the-fly applications.</p> <p>Y Like X (analogously).</p> <p>P End value of the signal parameter. As an unsigned 32-bit value. Allowed values: dependent on the selection made by set_vector_control (Ctrl parameter), identical with set_vector_control (Value parameter).</p>
Comments	<ul style="list-style-type: none"> If ""vector-controlled 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-controlled 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 Section "Vector-Defined Laser Control", page 205). There is no abs mechanism for P. P is clipped to the maximum allowed value. If para_jump_abs is used along with position-dependent 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 Section "Vector-Defined Laser Control", page 205.
RTC4→RTC6	<p>New command.</p> <p>In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for X and Y by 16. The allowed value ranges decrease accordingly. There is no RTC4 Compatibility Mode for the parameter P. The original RTC6 units must be used.</p> <p>Exception is Ctrl = 7 (Defocus): In RTC4 Compatibility Mode, with Ctrl = 7, the RTC6 multiplies the specified values for Value by 16. The allowed value ranges decrease accordingly.</p>
RTC5→RTC6	<p>Unchanged functionality.</p> <p>Exception is Ctrl = 7 (Defocus): In RTC5 Compatibility Mode, with Ctrl = 7, the RTC6 multiplies the specified values for Value by 16. The allowed value ranges decrease accordingly.</p>
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 the 3D image field . Simultaneously varies the signal parameter selected by set_vector_control linearly to the specified value.
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see select_cor_table), then para_jump_abs_3d has the same effect as para_jump_abs . However, split-up into Microsteps 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 Absolute x coordinate of the jump vector end point. In bits. As a signed 32-bit value. Allowed value range: [-268,435,456...+268,435,455]. Out-of-range values are clipped to the boundary values. The complete value range is only usable as a virtual Image field for example, for (enabled) Processing-on-the-fly applications.</p> <p>Y Like X (analogously).</p> <p>Z Like X (analogously), except Allowed value range: [-524,288...+524,287].</p> <p>P End value of the signal parameter. As an unsigned 32-bit value. Allowed values: dependent on the selection made by set_vector_control (Ctrl parameter), identical with set_vector_control (Value parameter).</p>
Comments	<ul style="list-style-type: none"> Except for the additional motion in the third dimension, para_jump_abs_3d functions similarly to the para_jump_abs command (see comments there). Further comments see jump_abs_3d.
RTC4→RTC6	<p>New command.</p> <p>In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for X, Y and Z coordinates by 16. The allowed value ranges decrease accordingly.</p> <p>There is no RTC4 Compatibility Mode for the parameter P. The original RTC6 units must be used. Exception is Ctrl = 7 (Defocus): In RTC4 Compatibility Mode, with Ctrl = 7, the RTC6 multiplies the specified values for Value by 16. The allowed value ranges decrease accordingly.</p>
RTC5→RTC6	<p>Unchanged functionality.</p> <p>Exception is Ctrl = 7 (Defocus): In RTC5 Compatibility Mode, with Ctrl = 7, the RTC6 multiplies the specified values for Value by 16. The allowed value ranges decrease accordingly.</p>
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 2D Image field . Simultaneously varies the signal parameter selected by set_vector_control linearly to the specified value.
Call	para_jump_rel(dx, dy, p)
Parameters	<p>dx Relative x coordinate of the jump vector end point. In bits. As a signed 32-bit value. Allowed value range: [-268,435,456...+268,435,455]. Out-of-range values are clipped to the boundary values. The complete value range is only usable as a virtual Image field for example, for (enabled) Processing-on-the-fly applications.</p> <p>dy Like dx (analogously).</p> <p>p End value of the signal parameter. As an unsigned 32-bit value. Allowed values: dependent on the selection made by set_vector_control (Ctrl parameter), identical with set_vector_control (Value parameter).</p>
Comments	<ul style="list-style-type: none"> The coordinates for the jump vector end point are to be supplied as relative coordinates with respect to the current position. Otherwise, para_jump_rel is identical to para_jump_abs (see comments there). p is not treated on a relative basis.
RTC4→RTC6	<p>New command.</p> <p>In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for dx and dy by 16. The allowed value range decreases accordingly. For the parameter p, see para_jump_abs.</p>
RTC5→RTC6	<p>Unchanged functionality.</p> <p>For the parameter p, see para_jump_abs.</p>
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 the 3D image field . Simultaneously varies the signal parameter selected by set_vector_control linearly to the specified value.
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see select_cor_table), then para_jump_rel_3d has the same effect as para_jump_rel . However, split-up into Microsteps 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 Relative x coordinate of the jump vector end point. In bits. As a signed 32-bit value. Allowed value range: [-268,435,456...+268,435,455]. Out-of-range values are clipped to the boundary values. The complete value range is only usable as a virtual Image field for example, for (enabled) Processing-on-the-fly applications.</p> <p>dy Like dx (analogously).</p> <p>dz Like dx (analogously), except Allowed value range: [-524,288...+524,287].</p> <p>P End value of the signal parameter. As an unsigned 32-bit value. Allowed values: dependent on the selection made by set_vector_control (Ctrl parameter), identical with set_vector_control (Value parameter).</p>
Comments	<ul style="list-style-type: none"> The coordinates for the jump vector end point are to be supplied as relative coordinates with respect to the current position. Otherwise, para_jump_rel_3d is identical to para_jump_abs_3d (see comments there). P is not treated on a relative basis.
RTC4→RTC6	<p>New command.</p> <p>In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for dx and dy by 16. The allowed value range decreases accordingly. For the parameter P, see para_jump_abs.</p>
RTC5→RTC6	<p>Unchanged functionality.</p> <p>For the parameter P, see para_jump_abs.</p>
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 signal for the specified number of external signal pulses (but for no longer than the specified time interval). Simultaneously varies the signal parameter selected by <code>set_vector_control</code> linearly 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 bits. As an unsigned 32-bit value. 1 bit equals $10\ \mu\text{s}$. Allowed value 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 value range: $0 \leq \text{Pulses} \leq 65,535$ 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 values: dependent on the selection made by <code>set_vector_control</code> (<code>Ctrl</code> parameter), identical with <code>set_vector_control</code> (<code>Value</code> parameter).</td> </tr> </table>	Period	Time interval. In bits. As an unsigned 32-bit value. 1 bit equals $10\ \mu\text{s}$. Allowed value range: $0 \leq \text{Period} \leq (2^{32}-1)$.	Pulses	Number of external signal pulses. As an unsigned 32-bit value. Allowed value range: $0 \leq \text{Pulses} \leq 65,535$ or larger (see comments below).	P	End value of the signal parameter. As an unsigned 32-bit value. Allowed values: dependent on the selection made by <code>set_vector_control</code> (<code>Ctrl</code> parameter), identical with <code>set_vector_control</code> (<code>Value</code> parameter).
Period	Time interval. In bits. As an unsigned 32-bit value. 1 bit equals $10\ \mu\text{s}$. Allowed value range: $0 \leq \text{Period} \leq (2^{32}-1)$.						
Pulses	Number of external signal pulses. As an unsigned 32-bit value. Allowed value range: $0 \leq \text{Pulses} \leq 65,535$ or larger (see comments below).						
P	End value of the signal parameter. As an unsigned 32-bit value. Allowed values: dependent on the selection made by <code>set_vector_control</code> (<code>Ctrl</code> parameter), identical with <code>set_vector_control</code> (<code>Value</code> parameter).						
Comments	<ul style="list-style-type: none"> • <code>para_laser_on_pulses_list</code> is useful for marking single dots, see Chapter 7.1.3 "Marking Single Dots", page 140. • If "vector-controlled laser control" has not been previously activated by <code>set_vector_control</code>, then <code>para_laser_on_pulses_list</code> behaves like <code>laser_on_pulses_list</code> and – if <code>Pulses > 65,535</code> – like <code>laser_on_list</code> (see comments there). The parameter <code>P</code> is then ignored. • If "vector-controlled laser control" is activated, then the signal parameter selected by <code>set_vector_control</code> is linearly varied from the last valid value to <code>P</code> within <code>para_laser_on_pulses_list</code> duration (<code>Period</code> $\times 10\ \mu\text{s}$), see Section "Vector-Defined Laser Control", page 205. • There is no abs mechanism for <code>P</code>. <code>P</code> is clipped to the maximum allowed value. This maximum value is $(2^{31}-1)$ or a lower value depending on what has been selected with <code>set_vector_control</code> (<code>Ctrl</code> parameter). • If <code>para_laser_on_pulses_list</code> is used along with position-dependent or speed-dependent laser control for the same control parameter, then the current value of <code>P</code> is used as the basis of the 100% value for laser control, see Section "Vector-Defined Laser Control", page 205. • If <code>Period = 0</code>, <code>para_laser_on_pulses_list</code> has no effect. Then <code>para_laser_on_pulses_list</code> is a short list command. • If $0 < \text{Period} \leq (2^{31}-1)$, then the <code>para_laser_on_pulses_list</code> duration is always <code>Period</code> clocks (that is, <code>Period</code> $\times 10\ \mu\text{s}$), 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 <code>para_laser_on_pulses_list</code> maximum duration is $(\text{Period} - 2^{31})$ clocks (that is, $(\text{Period} - 2^{31}) \times 10\ \mu\text{s}$). Here, however, <code>para_laser_on_pulses_list</code> terminates as soon as the specified number of external signal pulses has been detected. 						



Variable List Command	para_laser_on_pulses_list
RTC4→RTC6	New command. For the parameter P , see para_jump_abs .
RTC5→RTC6	Unchanged functionality. For the parameter P , see para_jump_abs .
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	laser_on_pulses_list , laser_on_list

Normal List Command	para_mark_abs
Function	Moves the laser focus at mark speed along a 2D vector from the current position to the specified position (absolute coordinate values) within a 2D Image field . Simultaneously varies the signal parameter selected by set_vector_control linearly to the specified value.
Call	<code>para_mark_abs(X, Y, P)</code>
Parameters	<p>X Absolute x coordinate of the mark vector end point. In bits. As a signed 32-bit value. Allowed value range: [-268,435,456...+268,435,455]. Out-of-range values are clipped to the boundary values. The complete value range is only usable as a virtual Image field for example, for (enabled) Processing-on-the-fly applications.</p> <p>Y Like X (analogously).</p> <p>P End value of the signal parameter. As an unsigned 32-bit value. Allowed values: dependent on the selection made by set_vector_control (Ctrl parameter), identical with set_vector_control (Value parameter).</p>
Comments	<ul style="list-style-type: none"> If "vector-controlled laser control" has not been previously activated by set_vector_control, then para_mark_abs behaves like mark_abs (see comments there). The parameter P is then ignored. If "vector-controlled laser control" is activated, then simultaneously with the laser focus' motion the signal parameter selected by set_vector_control is linearly varied from the last valid value to P see Section "Vector-Defined Laser Control", page 205. There is no abs mechanism for P. P is clipped to the maximum allowed value. • \leq OUT 618: For mark vector lengths = 0, no change of signal parameter P must be programmed: <ul style="list-style-type: none"> This change is not outputted The following [*]para[*] Command (only this one) produces incorrect signal parameter outputs • \geq OUT 619: For mark vector lengths = 0 where a change of signal parameter P is programmed, the following applies: <ul style="list-style-type: none"> These are handled as timed vectors with $T = 10 \mu s$ The end value of the signal parameter P is outputted If para_mark_abs is used along with position-dependent 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 Section "Vector-Defined Laser Control", page 205). [*]para_mark[*] commands generally do not take Sky Writing into account.



Normal List Command	para_mark_abs
RTC4→RTC6	New command. In RTC4 Compatibility Mode , the RTC6 multiplies the specified values for X and Y by 16. The allowed value range decreases accordingly. For the parameter P , see para_jump_abs .
RTC5→RTC6	Unchanged functionality. For the parameter P , see para_jump_abs .
Version info	Available as of DLL 600, OUT 600, RBF 600. Last change OUT 619: handling of mark vector lengths = 0.
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 mark speed along a 3D vector from the current position to the specified position (absolute coordinate values) within the 3D image field . Simultaneously varies the signal parameter selected by set_vector_control linearly to the specified value.
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see select_cor_table), then para_mark_abs_3d has the same effect as para_mark_abs . However, split-up into Microsteps is calculated like a 3D command and hence influences the effective mark speed in the xy plane.
Call	para_mark_abs_3d(X , Y , Z , P)
Parameters	<p>X Absolute x coordinate of the mark vector end point. In bits. As a signed 32-bit value. Allowed value range: [-268,435,456...+268,435,455]. Out-of-range values are clipped to the boundary values. The complete value range is only usable as a virtual Image field for example, for (enabled) Processing-on-the-fly applications.</p> <p>Y Like X (analogously).</p> <p>Z Like X (analogously), except Allowed value range: [-524,288...+524,287].</p> <p>P End value of the signal parameter. As an unsigned 32-bit value. Allowed values: dependent on the selection made by set_vector_control (Ctrl parameter), identical with set_vector_control (Value 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 comments there). Further comments see mark_abs_3d. [*]para_mark[*] commands generally do not take Sky Writing into account.
RTC4→RTC6	<p>New command.</p> <p>In RTC4 Compatibility Mode, the RTC6 multiplies the specified value for the x and y coordinates by 16. The allowed value range decreases accordingly.</p> <p>For the parameter P, see para_jump_abs.</p>
RTC5→RTC6	<p>Unchanged functionality.</p> <p>For the parameter P, see para_jump_abs.</p>
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 mark speed along a 2D vector from the current position to the specified position (relative coordinate values) within a 2D Image field . Simultaneously varies the signal parameter selected by set_vector_control linearly to the specified value.
Call	para_mark_rel(dx, dy, p)
Parameters	<p>dx Relative x coordinate of the mark vector end point. In bits. As a signed 32-bit value. Allowed value range: [-268,435,456...+268,435,455]. Out-of-range values are clipped to the boundary values. The complete value range is only usable as a virtual Image field for example, for (enabled) Processing-on-the-fly applications.</p> <p>dy Like dx (analogously).</p> <p>p End value of the signal parameter. As an unsigned 32-bit value. Allowed values: dependent on the selection made by set_vector_control (Ctrl parameter), identical with set_vector_control (Value parameter).</p>
Comments	<ul style="list-style-type: none"> The coordinates for the mark vector end point are to be supplied as relative coordinates with respect to the current position. Otherwise, para_mark_rel is identical to para_mark_abs (see comments there). p is not treated on a relative basis. [*]para_mark[*] commands generally do not take Sky Writing into account.
RTC4→RTC6	<p>New command.</p> <p>In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for dx and dy by 16. The allowed value range decreases accordingly. For the parameter p see para_jump_abs.</p>
RTC5→RTC6	<p>Unchanged functionality.</p> <p>For the parameter p, see para_jump_abs.</p>
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 mark speed along a 3D vector from the current position to the specified position (relative coordinate values) within the 3D image field . Simultaneously varies the signal parameter selected by set_vector_control linearly to the specified value.
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see select_cor_table), then para_mark_rel_3d has the same effect as para_mark_rel . However, split-up into Microsteps is calculated like a 3D command and hence influences the effective mark speed in the xy plane.
Call	para_mark_rel_3d(dx, dy, dz, P)
Parameters	<p>dx Relative x coordinate of the mark vector end point. In bits. As a signed 32-bit value. Allowed value range: [-268,435,456...+268,435,455]. Out-of-range values are clipped to the boundary values. The complete value range is only usable as a virtual Image field for example, for (enabled) Processing-on-the-fly applications.</p> <p>dy Like x (analogously).</p> <p>dz Like x (analogously), except Allowed value range: [-524,288...+524,287].</p> <p>P End value of the signal parameter. As an unsigned 32-bit value. Allowed values: dependent on the selection made by set_vector_control (Ctrl parameter), identical with set_vector_control (Value parameter).</p>
Comments	<ul style="list-style-type: none"> The coordinates for the mark vector end point are to be supplied as relative coordinates with respect to the current position. Otherwise, para_mark_rel_3d is identical to para_mark_abs_3d (see comments there). P is not treated on a relative basis. [*]para_mark[*] commands generally do not take Sky Writing into account.
RTC4→RTC6	New command. In RTC4 Compatibility Mode , the RTC6 multiplies the specified value for dx and dy by 16. The allowed value range decreases accordingly. For the parameter P, see para_jump_abs .
RTC5→RTC6	Unchanged functionality. For the parameter P, see para_jump_abs .
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 2D Image field .
Restriction	If the Option Processing-on-the-fly is not enabled, then park_position functions like a normal jump_abs .
Call	<code>park_position(Mode, X, Y)</code>
Parameters	<p>Mode As an unsigned 32-bit value.</p> <p>= 0: X and Y are interpreted as park position coordinates in the real Image field. Allowed value range: [-524,288...+524,287]. 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 value range: [-268,435,456...+268,435,455]. 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 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 clipped to the real Image field's boundaries.</p> <p>X Absolute x coordinate of the jump vector end point. In bits. As a signed 32-bit value. Allowed value range: see above. Out-of-range values are clipped to the boundary values.</p> <p>Y Like X (analogously).</p>
Comments	<ul style="list-style-type: none"> • park_position 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 forwarding motion (prior to list interruption by wait_for_encoder etc.), see Chapter 8.6.7 "Synchronizing Processing-on-the-fly Applications", page 251. 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).



Variable List Command	park_position
Comments (cont'd)	<ul style="list-style-type: none"> If the laser focus has been already previously moved to a safe park position, then park_position is a short list command with no effect. park_position switches off the Signals for "Laser Active" Operation after a LaserOff Delay, but does not activate a scanner delay afterward.
RTC4→RTC6	<p>New command.</p> <p>In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for X and Y by 16. The allowed value range decreases accordingly.</p>
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	park_return

Variable List Command	park_position_1_axis
Function	" Fly Extension " Command: Moves the output position for 1 Axis for intermediate parking with jump speed from the current to the specified absolute position in the 2D Image field .
Restriction	If the Option Processing-on-the-fly is not enabled, then park_position_1_axis only carries-out the jump to the specified new output position.
Call	<code>park_position_1_axis(Mode, Axis, ParkPos)</code>
Parameters	<p>Mode = 0: Intermediate parking position. Absolute coordinate in the real Image field. Processing-on-the-fly is switched off at the specified Axis.</p> <p>> 0: Intermediate parking position. Absolute coordinate in the virtual Image field. Processing-on-the-fly remains switched on at the specified Axis.</p> <p>As an unsigned 32-bit value.</p> <p>Axis Axis from Table 3, page 258.</p> <p>As an unsigned 32-bit value.</p> <p>Allowed values: 1...2.</p> <p>ParkPos Absolute axis coordinate of the new output position. In bits.</p> <p>As a signed 32-bit value.</p> <p>Out-of-range values are clipped to the boundary values.</p>
Comments	<ul style="list-style-type: none"> Being an "Fly Extension" Command, park_position_1_axis must not be used mixed with "Classic" Processing-on-the-fly commands (see Footnote, page 241). See Chapter 8.6 "Processing-on-the-fly", page 241 and Section ""Fly Extension" Commands", page 258. See also comments on park_position. To jump back, park_return_1_axis can be used. With an unallowed parameter value, park_position_1_axis is replaced by a list_nop (get_last_error return code RTC6_PARAM_ERROR).
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 617, OUT 617, RBF 623.
References	park_position_2_axes



Variable List Command	park_position_2_axes
Function	"Fly Extension" Command: Moves the output positions for 2 Axes for intermediate parking with jump speed from the current to the specified absolute position in the 2D Image field.
Restriction	If the Option Processing-on-the-fly is not enabled, then park_position_2_axes only carries-out the jump to the specified new output position.
Call	<code>park_position_2_axes(Mode, ParkPosX, ParkPosY)</code>
Parameters	<p>Mode = 0: Intermediate parking position. Absolute coordinate in the real Image field. Processing-on-the-fly is switched off at the specified Axes.</p> <p>> 0: Intermediate parking position. Absolute coordinate in the virtual Image field. Processing-on-the-fly remains switched on at the specified Axes.</p> <p>As an unsigned 32-bit value.</p> <p>ParkPosX Absolute axis coordinate of the new output position. In bits. As a signed 32-bit value. Out-of-range values are clipped to the boundary values.</p> <p>ParkPosY Like ParkPosX.</p>
Comments	<ul style="list-style-type: none"> Being an "Fly Extension" Command, park_position_2_axes must not be used mixed with "Classic" Processing-on-the-fly commands (see Footnote, page 241). See Chapter 8.6 "Processing-on-the-fly", page 241 and Section "Fly Extension" Commands, page 258. See also comments on park_position. To jump back, park_return_2_axes can be used. With an unallowed parameter value, park_position_2_axes is replaced by a list_nop (get_last_error return code RTC6_PARAM_ERROR). The following command calls are executed in the same way: <ul style="list-style-type: none"> - <code>park_position_2_axes(Mode, ParkPosX, ParkPosY) =</code> <code>{</code> <code> park_position_1_axis(Mode, 1, ParkPosX);</code> <code> park_position_1_axis(Mode, 2, ParkPosY);</code> <code>}</code>
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 617, OUT 617, RBF 623.
References	park_position_1_axis

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 2D Image field .
Restriction	If the Option Processing-on-the-fly is not enabled, then park_return functions as a normal Jump command . If the laser focus is not currently in a park position, then park_return is a short list command with no effect (see below).
Call	<code>park_return(Mode, X, Y)</code>
Parameters	<p>Mode As an unsigned 32-bit value. = 0: X and Y are ignored (see comments). > 0: X and Y are interpreted as return jump coordinates (see comments).</p> <p>X Absolute x coordinate of the jump vector end point in the virtual Image field. In bits. As a signed 32-bit value. Allowed value range: [-268,435,456...+268,435,455]. Out-of-range values are clipped to the boundary values.</p> <p>Y Like X (analogously).</p>
Comments	<ul style="list-style-type: none"> • park_return 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 forwarding motion (following list interruption by wait_for_encoder etc.), see Chapter 8.6.7 "Synchronizing Processing-on-the-fly Applications", page 251. See also comments at park_position. • If a set_fly_2d or set_fly_x/set_fly_y Processing-on-the-fly mode has been 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 has been attained by park_position, then park_return(Mode = 0) returns the galvanometer scanners to the most recent valid position before park_position has been called, whereas park_return(Mode = 1) moves them to the position specified with the command. When calculating the new position, the RTC6 takes the current Processing-on-the-fly correction into account. But if clipping to the boundaries of the virtual Image field occurs (for example, due to a too long forwarding 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.



Variable List Command	park_return
Comments (cont'd)	<ul style="list-style-type: none"> If the laser focus <i>has not been</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 has been 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 Section "Coordinate Transformations in the Virtual Image Field", page 168). After park_return, a Jump Delay is activated. park_return switches off the Signals for "Laser Active" Operation after a LaserOff Delay.
RTC4→RTC6	<p>New command.</p> <p>In RTC4 Compatibility Mode, the RTC6 multiplies the specified coordinate values by 16. The allowed value range decreases accordingly.</p>
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	park_position



Variable List Command	park_return_1_axis
Function	"Fly Extension" Command: Moves the output position for 1 Axis with jump speed from the intermediate parking position to the specified absolute position in the 2D Image field.
Restriction	If the Option Processing-on-the-fly is not enabled, then park_return_1_axis only carries-out the jump to the specified new output position.
Call	<code>park_return_1_axis(Mode, Axis, RetPos)</code>
Parameters	<p>Mode = 0: Return position is ignored. > 0: Return position. Absolute coordinate in the virtual Image field. As an unsigned 32-bit value.</p> <p>Axis Axis from Table 3, page 258. As an unsigned 32-bit value. Allowed values: 1...2.</p> <p>RetPos Absolute axis coordinate of the new output position. In bits. As a signed 32-bit value. Out-of-range values are clipped to the boundary values.</p>
Comments	<ul style="list-style-type: none"> Being an "Fly Extension" Command, park_return_1_axis must not be used mixed with "Classic" Processing-on-the-fly commands (see Footnote, page 241). See Chapter 8.6 "Processing-on-the-fly", page 241 and Section ""Fly Extension" Commands", page 258. See also comments on park_return. park_return_1_axis can be used to jump back to park_position_1_axis. If a Processing-on-the-fly correction has been previously deactivated at the specified axis by <code>park_position_1_axis(Mode = 0)</code> or <code>park_position_2_axes(Mode = 0)</code> then park_return_1_axis switches it back on again. With an unallowed parameter value, park_return_1_axis is replaced by a list_nop (get_last_error return code <code>RTC6_PARAM_ERROR</code>).
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 617, OUT 617, RBF 623.
References	park_return_2_axes



Variable List Command	park_return_2_axes
Function	" Fly Extension " Command: Moves the output position for 2 Axes with jump speed from the intermediate parking position to the specified absolute position in the 2D Image field .
Restriction	If the Option Processing-on-the-fly is not enabled, then park_return_2_axes only carries-out the jump to the specified new output positions.
Call	<code>park_return_2_axes(Mode, RetPosX, RetPosY)</code>
Parameters	<p>Mode = 0: Return positions are ignored. > 0: Return positions. Absolute coordinates in the virtual Image field. As an unsigned 32-bit value.</p> <p>RetPosX Absolute axes coordinates of the new output positions. In bits. As a signed 32-bit value. Out-of-range values are clipped to the boundary values.</p> <p>RetPosY Like RetPosX.</p>
Comments	<ul style="list-style-type: none"> Being an "Fly Extension" Command, park_return_2_axes must not be used mixed with "Classic" Processing-on-the-fly commands (see Footnote, page 241). See Chapter 8.6 "Processing-on-the-fly", page 241 and Section ""Fly Extension" Commands", page 258. See also comments on park_return. park_return_2_axes can be used to jump back to park_position_2_axes. If a Processing-on-the-fly correction has been previously deactivated at the specified axis by park_position_2_axes(Mode = 0) then park_position_2_axes switches it back on again. With an unallowed parameter value, park_return_2_axes is replaced by a list_nop (get_last_error return code RTC6_PARAM_ERROR). The following command calls are executed in the same way: <pre>- park_return_2_axes(Mode, RetPosX, RetPosY) = { park_return_1_axis(Mode, 1, RetPosX); park_return_1_axis(Mode, 2, RetPosY); }</pre>
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 617, OUT 617, RBF 623.
References	park_return_1_axis



Ctrl Command	pause_list
Function	Pauses execution of the list and disables the Signals for "Laser Active" Operation.
Call	<code>pause_list()</code>
Parameters	–
Comments	<ul style="list-style-type: none"> • pause_list is synonymous with stop_list. stop_list is often confused with stop_execution. Therefore, it is preferable to use pause_list. • If pause_list is called during execution of a list, then the Signals for "Laser Active" Operation are suppressed (the signals are set to their respective "Off" level) and keeps the scan system in the most recently defined state – even if in the middle of a split-up into Microsteps. The PAUSED list execution status (queryable by get_status) is set, but the BUSY list execution status is left unchanged. Continuation by execute_list_pos or release_wait or by an External Start is not possible. However, stop_execution or an External Stop is possible. restart_list, stop_execution or an External Stop ends suppression of the start. • If processing of a list should be continued, then restart_list must be used. After a subsequent restart_list, the scan system resumes the planned movements (of the current command) and the laser control signals are released again (in general, an interrupted marking cannot be continued without a disruption in the marking result). The PAUSED list execution status is then reset (here too, BUSY list execution status remains unchanged). • If, during calling of pause_list, no list is currently executing (BUSY list execution status not set) or a list has already been halted by pause_list or set_wait (PAUSED list execution status set), then pause_list is ignored (get_last_error return code RTC6_BUSY; the laser is then already off).
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	restart_list , set_wait



Ctrl Command	periodic_toggle												
Function	Like periodic_toggle_list , but a control command.												
Call	<code>periodic_toggle(Port, Mask, P1, P2, Count, Start)</code>												
Parameters	<table> <tr> <td>Port</td> <td>Like periodic_toggle_list.</td> </tr> <tr> <td>Mask</td> <td>Like periodic_toggle_list.</td> </tr> <tr> <td>P1</td> <td>Like periodic_toggle_list.</td> </tr> <tr> <td>P2</td> <td>Like periodic_toggle_list.</td> </tr> <tr> <td>Count</td> <td>Like periodic_toggle_list.</td> </tr> <tr> <td>Start</td> <td>Like periodic_toggle_list.</td> </tr> </table>	Port	Like periodic_toggle_list .	Mask	Like periodic_toggle_list .	P1	Like periodic_toggle_list .	P2	Like periodic_toggle_list .	Count	Like periodic_toggle_list .	Start	Like periodic_toggle_list .
Port	Like periodic_toggle_list .												
Mask	Like periodic_toggle_list .												
P1	Like periodic_toggle_list .												
P2	Like periodic_toggle_list .												
Count	Like periodic_toggle_list .												
Start	Like 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 <code>periodic_toggle</code> is not executed (<code>get_last_error</code> return code <code>RTC6_PARAM_ERROR</code>). • See also Chapter 9.4 "Periodical I/O Signals", page 300. 												
RTC4→RTC6	New command.												
RTC5→RTC6	Unchanged functionality.												
Version info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 605, OUT 605: endless toggling with <code>Count</code> = $2^{32}-1$.												
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	<table> <tr> <td>Port</td> <td>Output port (0...4, as with set_port_default). As an unsigned 32-bit value.</td> </tr> <tr> <td>Mask</td> <td>Defines the bits to be toggled, (the maximum value depends on the port). 1 = bit is toggled. 0 = bit remains unchanged. As an unsigned 32-bit value.</td> </tr> <tr> <td>P1</td> <td>Duration of the toggled signal in [10 μs] (> 0, 16 bit max.). As an unsigned 32-bit value.</td> </tr> <tr> <td>P2</td> <td>Duration of the toggled signal in [10 μs] (> 0, 16 bit max.). As an unsigned 32-bit value.</td> </tr> <tr> <td>Count</td> <td>Number of P1–P2 period repetitions. As an unsigned 32-bit value.</td> </tr> <tr> <td>Start</td> <td>Duration until the first toggling starts in [10 μs]. As an unsigned 32-bit value.</td> </tr> </table>	Port	Output port (0...4, as with set_port_default). As an unsigned 32-bit value.	Mask	Defines the bits to be toggled, (the maximum value depends on the port). 1 = bit is toggled. 0 = bit remains unchanged. As an unsigned 32-bit value.	P1	Duration of the toggled signal in [10 μ s] (> 0, 16 bit max.). As an unsigned 32-bit value.	P2	Duration of the toggled signal in [10 μ s] (> 0, 16 bit max.). As an unsigned 32-bit value.	Count	Number of P1–P2 period repetitions. As an unsigned 32-bit value.	Start	Duration until the first toggling starts in [10 μ s]. As an unsigned 32-bit value.
Port	Output port (0...4, as with set_port_default). As an unsigned 32-bit value.												
Mask	Defines the bits to be toggled, (the maximum value depends on the port). 1 = bit is toggled. 0 = bit remains unchanged. As an unsigned 32-bit value.												
P1	Duration of the toggled signal in [10 μ s] (> 0, 16 bit max.). As an unsigned 32-bit value.												
P2	Duration of the toggled signal in [10 μ s] (> 0, 16 bit max.). As an unsigned 32-bit value.												
Count	Number of P1–P2 period repetitions. As an unsigned 32-bit value.												
Start	Duration until the first toggling starts in [10 μ s]. As an unsigned 32-bit value.												
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 <code>periodic_toggle_list</code> is replaced by a <code>list_nop</code> and a <code>get_last_error</code> return code <code>RTC6_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 cycles. Then they are toggled once again and kept stable for $P2 \times 10 \mu$s clock cycles. 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 <code>periodic_toggle_list</code>. The first toggling occurs after <code>Start</code> $\times 10 \mu$s clock cycles. Toggling is repeated Count-times. Count = 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...65,535]. <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 <code>set_end_of_list</code>, <code>stop_execution</code> or external /STOP the periodical signals continue. <code>periodic_toggle_list</code> toggles endless with <code>Count</code> = $2^{32}-1$. See also Chapter 9.4 "Periodical I/O Signals", page 300. 												



Undelayed Short List Command	periodic_toggle_list
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 605, OUT 605: endless toggling with Count = $2^{32}-1$.
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 is fully executed until the next set_end_of_list is encountered. start_loop must be called prior to quit_loop. Otherwise, quit_loop has no effect.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	start_loop

Undelayed Short List Command	range_checking	
Function	Defines an emergency action, if a galvanometer scanner exceeds its position limit.	
Call	range_checking(HeadNo, Mode, Data)	
Parameters	HeadNo	<p>The scan system to be monitored. As an unsigned 32-bit value.</p> <p>0: No monitoring (default after load_program_file). 1: First scan head is monitored. 2: Second scan head is monitored. 3: Both <i>scan heads</i> are monitored.</p> <p>Only the 2 least significant bits are evaluated.</p>
	Mode	<p>Action to take. As an unsigned 32-bit value.</p> <p>0: The Signals for "Laser Active" Operation are suppressed immediately. List execution continues. 1: The Signals for "Laser Active" Operation are switched-off immediately. List execution is stopped immediately (as with stop_execution or /STOP). 2: A simulate_ext_stop is passed on to all slave boards.</p>
	Data	<p>Data type chosen to be monitored. As an unsigned 32-bit value.</p> <p>0: Sample values. Like set_trigger signal types 7...9. 1: Transformed control values. Like set_trigger signal types 25...30. 2: Corrected control values. Like set_trigger signal types 10...15. 3: Actual output values. Like set_trigger signal types 20...23. 4: Actual galvanometer scanner position (only with <i>iDRI</i>VE systems, <i>varioSCAN</i> is internally connected to an x axis). Like set_trigger signal types [1, 2 and 4] or [4, 5 and 1]. 5: Actual galvanometer scanner position (only with <i>iDRI</i>VE systems, <i>varioSCAN</i> is internally connected to a y axis). Like set_trigger signal types [1, 2 and 5] or [4, 5 and 2].</p>
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). 	

Undelayed Short List Command	range_checking
Comments (cont'd)	<ul style="list-style-type: none"> 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_x_overflow/if_fly_y_overflow/if_fly_z_overflow and if_not_fly_x_overflow/if_not_fly_y_overflow/if_not_fly_z_overflow. They do not trigger any activities. Processing-on-the-fly applications use the data type <code>Data = 0</code> (sample values) for customer-specific range limits. Inconsistent error messages and switch-offs may occur if used simultaneously with range_checking data types <code>Data > 0</code>. If the laser has been switched-off with <code>Mode = 0</code> and the fault condition is gone then the laser is <i>not</i> switched on until the next Mark command. The laser is <i>not</i> switched-on right in the middle of a currently executing Mark command, not even in the middle of a Polyline. The range_checking monitoring is active without Processing-on-the-fly application. <code>Data > 5</code> causes a get_last_error return code RTC6_PARAM_ERROR. In this case range_checking is sent to the RTC6 board as list_nop. <code>Data=2</code> and <code>Data=3</code> have the identical effect for the z axis. For <code>Data = 4</code> and <code>Data = 5</code> users must define the set position as the to-be-returned data type by the scan system. A simultaneous use of a speed-dependent laser control with <code>Mode = 2</code> (see set_auto_laser_control) is not possible. <code>Data = 4</code> and <code>Data = 5</code> only differ in regards to the axis onto an varioSCAN is connected. For 2D systems without varioSCAN both selections have the identical effect. <code>Data = 4</code> and <code>Data = 5</code> produce nonsensical switch-offs if an intelliSCAN is connected but is either not switched-on or a actual position has not been set as feedback.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 615: <code>Mode = 2</code> .
References	set_fly_limits , set_fly_limits_z



Ctrl Command	read_abc_from_file	
Function	Like read_abc_from_file_20b . However, for a focus length value l in RTC4 compatibility range $[-32,768\dots+32,767]$.	
Call	ErrorNo = read_abc_from_file(Name, &A, &B, &C)	
Result	ErrorNo Like read_abc_from_file_20b .	
Parameters	Name Like read_abc_from_file_20b .	
Returned parameter values	A B C	Coefficients of the parabolic function $z_{out} = A + Bl + Cl^2$ which is used for calculating the Z output values (focus length value l in the RTC4 compatibility range $[-32,768\dots+32,767]$). As 64-bit IEEE floating point values.
Comments	<ul style="list-style-type: none"> Like read_abc_from_file_20b. read_abc_from_file(&A, &B, &C) is synonymous with read_abc_from_file_20b(&A, &B, &C × 1/16). 	
RTC4→RTC6	New command.	
RTC5→RTC6	Unchanged functionality.	
Version info	Available as of DLL 600, OUT 600, RBF 600.	
References	wwrite_abc_to_file	

Ctrl Command	read_abc_from_file_20b											
Function	Reads the ABC values directly from a specified correction file on the PC. For a focus length value <i>l</i> in the RTC6 20-bit range [-524,288...+524,287].											
Call	ErrorNo = read_abc_from_file_20b(<i>Name</i> , <i>&A</i> , <i>&B</i> , <i>&C</i>)											
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>File error (corrupt or incomplete).</td> </tr> <tr> <td>2</td> <td>Memory error (RTC6 DLL-internal, Windows working memory).</td> </tr> <tr> <td>3</td> <td>File-open error (empty string, file not found etc.).</td> </tr> </table>		ErrorNo	Error code. As an unsigned 32-bit value.	0	No error.	1	File error (corrupt or incomplete).	2	Memory error (RTC6 DLL-internal, Windows working memory).	3	File-open error (empty string, file not found etc.).
ErrorNo	Error code. As an unsigned 32-bit value.											
0	No error.											
1	File error (corrupt or incomplete).											
2	Memory error (RTC6 DLL-internal, Windows working memory).											
3	File-open error (empty string, file not found etc.).											
Parameters	<i>Name</i>	Name of the correction file. As a pointer to a \0-terminated ANSI string.										
Returned parameter values	<i>A</i> <i>B</i> <i>C</i>	Coefficients of the parabolic function $z_{out} = A + Bl + C/l^2$ which is used for calculating the Z output values (focus length value <i>l</i> in the RTC6 20-bit range [-524,288...+524,287]). As 64-bit IEEE floating point values.										
Comments	<ul style="list-style-type: none"> • <code>read_abc_from_file_20b</code> is available even without explicit access rights to a specific RTC6 board. • <code>read_abc_from_file_20b</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 129, are neither generated nor altered by <code>read_abc_from_file_20b</code>. • <code>read_abc_from_file_20b(&A, &B, &C)</code> is synonymous with <code>read_abc_from_file(&A x 16, &B, &C x 1/16)</code>. 											
RTC4→RTC6	New command.											
RTC5→RTC6	New command.											
Version info	Available as of DLL 631 , OUT 632 .											
References	write_abc_to_file_20b											

Ctrl Command	read_analog_in																												
Function	<p>Reads in the analog input values:</p> <ul style="list-style-type: none"> • ANALOG IN0 and ANALOG IN1 at the "McBSP/ANALOG" socket connector of the RTC6 PCIe Board • ANALOG IN0, and ANALOG IN1 at the SPI/ANA/UART Socket Connector of the RTC6 Ethernet Board <p>Returns them as 12-bit digital values.</p>																												
Call	<code>AnalogValue = read_analog_in()</code>																												
Result	<p>As an unsigned 32-bit value.</p> <table> <tr> <td>Bit #0</td> <td>ANALOG IN0 input value as 12-bit digital value.</td> </tr> <tr> <td>...</td> <td></td> </tr> <tr> <td>Bit #11</td> <td></td> </tr> <tr> <td>Bit #12</td> <td>= 0. Channel number.</td> </tr> <tr> <td>Bit #13</td> <td>= 0.</td> </tr> <tr> <td>Bit #14</td> <td>= 0.</td> </tr> <tr> <td>Bit #15</td> <td>Old bit for ANALOG IN0.</td> </tr> <tr> <td>Bit #16</td> <td>ANALOG IN1 input value as 12-bit digital value.</td> </tr> <tr> <td>...</td> <td></td> </tr> <tr> <td>Bit #27</td> <td></td> </tr> <tr> <td>Bit #28</td> <td>= 1. Channel number.</td> </tr> <tr> <td>Bit #29</td> <td>= 0.</td> </tr> <tr> <td>Bit #30</td> <td>= 0.</td> </tr> <tr> <td>Bit #31</td> <td>Old bit for ANALOG IN1.</td> </tr> </table>	Bit #0	ANALOG IN0 input value as 12-bit digital value.	...		Bit #11		Bit #12	= 0. Channel number.	Bit #13	= 0.	Bit #14	= 0.	Bit #15	Old bit for ANALOG IN0 .	Bit #16	ANALOG IN1 input value as 12-bit digital value.	...		Bit #27		Bit #28	= 1. Channel number.	Bit #29	= 0.	Bit #30	= 0.	Bit #31	Old bit for ANALOG IN1 .
Bit #0	ANALOG IN0 input value as 12-bit digital value.																												
...																													
Bit #11																													
Bit #12	= 0. Channel number.																												
Bit #13	= 0.																												
Bit #14	= 0.																												
Bit #15	Old bit for ANALOG IN0 .																												
Bit #16	ANALOG IN1 input value as 12-bit digital value.																												
...																													
Bit #27																													
Bit #28	= 1. Channel number.																												
Bit #29	= 0.																												
Bit #30	= 0.																												
Bit #31	Old bit for ANALOG IN1 .																												
Comments	<ul style="list-style-type: none"> • See also Section "Analog Input Ports", page 85. • read_analog_in sets the old bits (Bit #15 and Bit #31) to 1 after reading. As soon new data are available at the corresponding analog input they are automatically set to 0. • The analog input values (ANALOG IN0 and ANALOG IN1) can be recorded by set_trigger/set_trigger4 (signal 54). However, the old bits (Bit #15 and Bit #31) are not recorded. The format of the data is ((ANALOG IN1 << 16) + ANALOG IN0). 																												
RTC4→RTC6	New command.																												
RTC5→RTC6	Unchanged functionality.																												
Version info	Available as of DLL 600, OUT 600, RBF 600.																												
References	set_trigger, set_trigger4																												



Ctrl Command	read_encoder
Function	Returns the counts of the two RTC6 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>Encoder0_0 Count. As a pointer to a signed 32-bit value. For parameter "Encoder_n_m":</p> <ul style="list-style-type: none"> • <i>n</i> is the number of the encoder counter ("Encoder0", "Encoder1"). • <i>m</i> is the number of the storage position (parameter <i>Pos</i> of store_encoder). <p>Encoder1_0 Like Encoder0_0 (analogously).</p> <p>Encoder0_1 Like Encoder0_0 (analogously).</p> <p>Encoder1_1 Like Encoder0_0 (analogously).</p>
Comments	<ul style="list-style-type: none"> • See also Chapter 8.6 "Processing-on-the-fly", page 241 and Chapter 9.3.3 "Synchronization by Encoder Signals", page 297. • If the workpiece movement is registered with an incremental encoder: <ul style="list-style-type: none"> – Encoder counter "Encoder0" is triggered by the signals at encoder input port ENCODER X – Encoder counter "Encoder1" is triggered by the signals at encoder input port ENCODER Y • If an encoder simulation has been started by simulate_encoder, the encoder counters are triggered by an internal periodic 1 MHz clock signal. • For storage positions in which counts were not previously stored by store_encoder, the value 0 is returned (initialized value).
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	store_encoder, get_encoder, set_fly_x, set_fly_y, set_fly_rot, wait_for_encoder

Ctrl Command	read_image_eth																								
Function	Standalone Functionality : Reads out data for automatic booting from the NAND memory and saves it as a binary file on the PC.																								
Prerequisite	RTC6 Software Package \geq V1.7.0 and BIOS-ETH \geq 26.																								
Call	<code>Result = read_image_eth(Name)</code>																								
Parameters	<table> <tr> <td>Name</td> <td>Name of the binary file. As a pointer to a \0-terminated ANSI string.</td> </tr> </table>	Name	Name of the binary file. As a pointer to a \0-terminated ANSI string.																						
Name	Name of the binary file. As a pointer to a \0-terminated ANSI string.																								
Result	<table> <tr> <td>Result</td> <td>Error code. As an unsigned 32-bit value.</td> </tr> <tr> <td></td> <td>Value Description</td> </tr> <tr> <td>0</td> <td>No error.</td> </tr> <tr> <td>1</td> <td>No RTC6 Ethernet Board.</td> </tr> <tr> <td>2</td> <td>Empty string has been specified as file name.</td> </tr> <tr> <td>3</td> <td>Access denied.</td> </tr> <tr> <td>4</td> <td>RTC6 Ethernet Board is BUSY list execution status.</td> </tr> <tr> <td>5</td> <td>Timeout error (RTC6 Ethernet Board does not respond).</td> </tr> <tr> <td>6</td> <td>File cannot be opened.</td> </tr> <tr> <td>7</td> <td>Ethernet error, see eth_get_last_error.</td> </tr> <tr> <td>8</td> <td>NAND memory error.</td> </tr> <tr> <td>9</td> <td>Aborted, see eth_get_last_error.</td> </tr> </table>	Result	Error code. As an unsigned 32-bit value.		Value Description	0	No error.	1	No RTC6 Ethernet Board.	2	Empty string has been specified as file name.	3	Access denied.	4	RTC6 Ethernet Board is BUSY list execution status.	5	Timeout error (RTC6 Ethernet Board does not respond).	6	File cannot be opened.	7	Ethernet error, see eth_get_last_error .	8	NAND memory error.	9	Aborted, see eth_get_last_error .
Result	Error code. As an unsigned 32-bit value.																								
	Value Description																								
0	No error.																								
1	No RTC6 Ethernet Board.																								
2	Empty string has been specified as file name.																								
3	Access denied.																								
4	RTC6 Ethernet Board is BUSY list execution status.																								
5	Timeout error (RTC6 Ethernet Board does not respond).																								
6	File cannot be opened.																								
7	Ethernet error, see eth_get_last_error .																								
8	NAND memory error.																								
9	Aborted, see eth_get_last_error .																								
Comments	<ul style="list-style-type: none"> • read_image_eth is not executed (get_last_error return code RTC6_BUSY), if: <ul style="list-style-type: none"> – the BUSY list execution status is set – the INTERNAL-BUSY list execution status is set • If the <code>Name</code> cannot be opened, a get_last_error return code RTC6_PARAM_ERROR is generated. • The content of an already existing binary file is deleted. • During the execution of read_image_eth the 10 μs clock cycle of the DSP is interrupted for up to 2 minutes. • read_image_eth is only allowed with RTC6 Ethernet Boards. Otherwise, a get_last_error return code RTC6_TYPE_REJECTED is generated. • See Chapter 16.7 "Standalone Functionality", page 890. 																								
RTC4→RTC6	New command.																								
RTC5→RTC6	New command.																								
Version info	Available as of DLL 618, OUT 618, RBF 623.																								
References	wwrite_image_eth, store_program																								



Ctrl Command	read_io_port
Function	Returns the current state of the 16-bit digital input port on the EXTENSION 1 socket connector.
Call	IOPort = read_io_port()
Result	16-bit value (DIGITAL IN0...DIGITAL IN15). As an unsigned 32-bit value.
Comments	<ul style="list-style-type: none">• See Chapter 9.2.1 "16-Bit Digital Input Port", page 287.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 port; the galvanometer scanner set position and time value that has been current at the time of reading).
Call	currentIndex = <code>read_io_port_buffer(Index, &Value, &XPos, &YPos, &Time)</code>
Parameters	<p>Index Number of the to-be-returned entry in the IOPort buffer. As an unsigned 32-bit value. Only the 12 least significant bits are evaluated (the IOPort buffer only has 4,096 entries). Therefore, the user program can use a continuous counter for the index.</p>
Returned parameter values	<p>Value 16-bit value. As a pointer to an unsigned 32-bit value.</p> <p>XPos Set position of the galvanometer scanner (x value). As a pointer to a signed 32-bit value.</p> <p>YPos Set position of the galvanometer scanner (y value). As a pointer to a signed 32-bit value.</p> <p>Time Time. In seconds. As a pointer to an unsigned 32-bit value.</p>
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 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→RTC6	New command.
RTC5→RTC6	Unchanged functionality. Larger IOPort buffer.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 port. At the same time, the current xy set positions and <i>relative</i> time in seconds are stored.
Call	<code>read_io_port_list()</code>
Comments	<ul style="list-style-type: none"> • read_io_port_list 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 4,096 entries and is circularly organized. It always stores the 4,096 most recent entries and overwrites older entries without warning. The incremental Index starts after a reset (program start) at Index 0, and after passing 4,095 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 Port", page 287.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality. Larger IOPort buffer.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	read_io_port_buffer , read_io_port

Ctrl Command	read_mcbsp
Function	Returns the most recent input value that has been fully copied to an internal memory location by the McBSP interface.
Call	<code>mcbsp_value = read_mcbsp(No)</code>
Parameters	<p>No Number of the internal memory location whose input value should be queried. As an unsigned 32-bit value.</p> <p>0 Memory location for Processing-on-the-fly applications and for set_mcbsp_in input with coding Bit #31 = 0.</p> <p>1 Memory locations for Online Positioning and for set_mcbsp_in input.</p> <p>2 Wie 1.</p> <p>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 2 least significant bits are evaluated.</p>
Result	Input value. As a signed 32-bit value.
Comments	<ul style="list-style-type: none"> For information on the McBSP interface and the related memory locations, see also Chapter 4.6.6 "McBSP/ANALOG Socket Connector", page 83. The interpretation as <ul style="list-style-type: none"> one signed or unsigned 32-bit data word two signed 16-bit data words two signed 15-bit data words is the responsibility of the user: <ul style="list-style-type: none"> For Processing-on-the-fly applications ($No = 0$) see Chapter 8.6 "Processing-on-the-fly", page 241 and Chapter 9.3.4 "Synchronization and Online Positioning by McBSP Signals", page 299. For "Local Online Positioning" ($No = 1\dots 2$) see Chapter 8.3.1 ""Local Online Positioning"", page 227. For "Global Online Positioning" ($No = 1\dots 2$) see Chapter 8.3.2 ""Global Online Positioning"", page 230. For set_mcbsp_in input ($No = 0\dots 3$) see Section "Correction via McBSP Interface with Additional McBSP Input", page 245 and Section "Correction via McBSP Interface with Additional McBSP Input", page 247. For set_multi_mcbsp_in, see page 693. After load_program_file, the input values at the McBSP interface are transferred to location 0 (default) even if no Processing-on-the-fly correction is activated. The McBSP interface ignores the first FrameSync signal after a load_program_file or mcbsp_init. That is, data provided is not transmitted, see page 85.

Ctrl Command	read_mcbsp
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	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 , set_mcbsp_global_x , set_mcbsp_global_y , set_mcbsp_global_rot , set_mcbsp_global_matrix

Ctrl Command	read_multi_mcbsp
Function	Queries the McBSP multi transmission input value that has been most recently type-sorted and stored.
Call	<code>mcbsp_value = read_multi_mcbsp(No)</code>
Parameters	No Number of the type-sorted internal memory location. No corresponds to coding Bit #0...Bit #2 of the McBSP multi input. As an unsigned 32-bit value. 0 x coordinate of the Processing-on-the-fly application. 1 y coordinate of the Processing-on-the-fly application. 2 z coordinate of the Processing-on-the-fly application. 3 Laser power P. 4 Extra parameter E1. 5 Extra parameter E2. 6 Extra parameter E3. 7 Extra parameter E4. Only the 3 least significant bits are evaluated.
Result	Memory value. As a signed 32-bit value.
Comments	<ul style="list-style-type: none"> You must have previously activated and used McBSP multi transmission by set_multi_mcbsp_in or set_multi_mcbsp_in_list. Otherwise, default or old values are returned.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_multi_mcbsp_in , set_multi_mcbsp_in_list

Ctrl Command	read_status																				
Function	Returns the RTC6 list status, see Chapter 6.4.2 "List Status", page 106 .																				
Call	Status = <code>read_status()</code>																				
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", that is, that all following list commands are stored in "List 1". LOAD1 is set if the input pointer is set into "List 1" (for example, by set_start_list_pos). LOAD1 is reset if a set_end_of_list is written to "List 1" (READY1 set) or if LOAD2 is set (for example, 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", that is, that all following list commands are stored in "List 2". LOAD2 is set if the input pointer is set into "List 2" (for example, by set_start_list_pos). LOAD2 is reset if a set_end_of_list is written to "List 2" (READY2 set) or if LOAD1 is set (for example, 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 has been written to "List 1". READY1 is reset if LOAD1 is newly set (for example, 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 has been written to "List 2". READY2 is reset if LOAD2 is newly set (for example, 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" has been started). BUSY1 is set by starting execution of "List 1" (for example, 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 (for example, list_jump_pos), if "List 2" is started (for example, 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", that is, that all following list commands are stored in "List 1". LOAD1 is set if the input pointer is set into "List 1" (for example, by set_start_list_pos). LOAD1 is reset if a set_end_of_list is written to "List 1" (READY1 set) or if LOAD2 is set (for example, by set_start_list_pos).	Bit #1	LOAD2	= 1: Indicates that the input pointer is currently in "List 2", that is, that all following list commands are stored in "List 2". LOAD2 is set if the input pointer is set into "List 2" (for example, by set_start_list_pos). LOAD2 is reset if a set_end_of_list is written to "List 2" (READY2 set) or if LOAD1 is set (for example, by set_start_list_pos).	Bit #2	READY1	= 1: Indicates that during the loading procedure a set_end_of_list has been written to "List 1". READY1 is reset if LOAD1 is newly set (for example, by set_start_list_pos).	Bit #3	READY2	= 1: Indicates that during the loading procedure a set_end_of_list has been written to "List 2". READY2 is reset if LOAD2 is newly set (for example, 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" has been started). BUSY1 is set by starting execution of "List 1" (for example, 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 (for example, list_jump_pos), if "List 2" is started (for example, 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", that is, that all following list commands are stored in "List 1". LOAD1 is set if the input pointer is set into "List 1" (for example, by set_start_list_pos). LOAD1 is reset if a set_end_of_list is written to "List 1" (READY1 set) or if LOAD2 is set (for example, by set_start_list_pos).																			
Bit #1	LOAD2	= 1: Indicates that the input pointer is currently in "List 2", that is, that all following list commands are stored in "List 2". LOAD2 is set if the input pointer is set into "List 2" (for example, by set_start_list_pos). LOAD2 is reset if a set_end_of_list is written to "List 2" (READY2 set) or if LOAD1 is set (for example, by set_start_list_pos).																			
Bit #2	READY1	= 1: Indicates that during the loading procedure a set_end_of_list has been written to "List 1". READY1 is reset if LOAD1 is newly set (for example, by set_start_list_pos).																			
Bit #3	READY2	= 1: Indicates that during the loading procedure a set_end_of_list has been written to "List 2". READY2 is reset if LOAD2 is newly set (for example, 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" has been started). BUSY1 is set by starting execution of "List 1" (for example, 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 (for example, list_jump_pos), if "List 2" is started (for example, 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" has been started). BUSY2 is set by starting execution of "List 2" (for example, by execute_list_pos) or by a list change to "List 2" (automatic list change or jump). BUSY2 is reset if set_end_of_list is executed in "List 2", if a jump into "List 1" is executed (for example, list_jump_pos), if "List 1" is started (for example, by execute_list_pos) or if stop_execution is executed.
	Bit #6	USED1	= 1:	Indicates that a set_end_of_list has been reached during processing of "List 1". USED1 is reset when LOAD1 is set (for example, by set_start_list_pos).
	Bit #7	USED2	= 1:	Indicates that a set_end_of_list has been reached during processing of "List 2". USED2 is reset when LOAD2 is set (for example, by set_start_list_pos).
	Bit #8		0	
	...			
	Bit #31			
Comments	<ul style="list-style-type: none"> When interpreting the status values returned by read_status, always take into account the programmed loading or execution processes of the lists. <p>Under some circumstances, the status values can be misleading, as illustrated by the following examples:</p> <ul style="list-style-type: none"> – Even during a loading process, the LOAD list status can already have been reset and the READY list status set if – after loading of a set_end_of_list into a list – further list commands are loaded into the same list. – The status values remain unchanged if a set_end_of_list is overwritten with another command (READY list status is not reset). – If – during a loading process – a set_end_of_list is processed at the same time in the same list, then the list is regarded as already processed (USED list status set), even though it is still newly loaded (USED list status has been then initially reset). – Even if a completely loaded command list (incl. set_end_of_list) has been stored, the READY list status of a list can be reset if the input pointer has been newly set (for example, by set_input_pointer) into the list. Then a list's USED list 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", for example, by list_jump_pos) during execution, the USED list status values of both lists (unlike their BUSY list execution 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 list memory 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 has been 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 Status values BUSY list execution status and PAUSED list execution status can be queried by get_status. To read the status signals from the <i>scan heads</i>, use get_head_status.
RTC4→RTC6	Basically unchanged functionality. However: Additional USED list status.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	get_status , get_head_status

Ctrl Command	read_user_data
Function	No function.
Comments	<ul style="list-style-type: none"> To date, read_user_data has no effect.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	send_user_data



Normal List Command	regulation3
Function	Category: List Commands for a Restricted User Group only. Varies laser frequency and pulse length depending on encoder frequency.
Call	regulation3(Fmax, Fmin)
Parameters	Fmax Max. encoder frequency. In Hz. Allowed value range: 10,000...1,000,000. As an unsigned 32-bit value.
	Fmin Switch-off frequency. In Hz. Allowed value range: 10...[Fmax / 10]. As an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> regulation3 is executed infinitely until list execution is stopped by stop_execution or an external /STOP. For encoder frequency < Fmin the laser is switched off. An encoder frequency > Fmax is clipped to Fmax. The value for the laser frequency results from the lookup table of set_laser_timing_table. The value for the pulse length results from the lookup table of set_duty_cycle_table.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 626, OUT 627.
References	set_duty_cycle_table , set_laser_timing_table

Ctrl Command	release_rtc
Function	Releases the specified RTC6 for use by other user programs.
Call	NoOfReleasedCard = <code>release_rtc(CardNo)</code>
Parameters	CardNo RTC6 DLL -internal number of the RTC6 board. As an unsigned 32-bit value.
Result	The returned value is <code>CardNo</code> if the user program has been still in possession of access rights for this board. Otherwise, 0 is returned. As an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> After the user program releases a board with <code>release_rtc</code>, it no longer has access rights for this board. The board can then be subsequently acquired by the same or another user program by <code>acquire_rtc</code> or <code>init_rtc6_dll</code>. If a board released by <code>release_rtc</code> has been 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 <code>select_rtc</code>. <code>release_rtc</code> is available even without explicit access rights for a particular RTC6 board, but <code>release_rtc</code> then has no effect (return value 0). <code>release_rtc</code> also has no effect (return value 0), if: <ul style="list-style-type: none"> CardNo exceeds the number of RTC6 boards found during initialization (see <code>rtc6_count_cards</code>) and No RTC6 Ethernet Board is entered at CardNo CardNo = 0 (real boards begin at 1) <code>release_rtc</code> is not available as a multi-board command.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	acquire_rtc , init_rtc6_dll

Ctrl Command	release_wait
Function	Resumes processing of a list that has been interrupted by set_wait .
Call	<code>release_wait()</code>
Comments	<ul style="list-style-type: none"> • release_wait is only executed if the RTC6 is actually in a wait state (hence if a break point has been previously reached and list processing has been interrupted; the PAUSED list execution status is then set but <i>not</i> the BUSY list execution status). Otherwise, release_wait is ignored (get_last_error return code RTC6_BUSY). • By release_wait, the PAUSED list execution status (queriable with get_status) is reset and the BUSY list execution status is newly set, see also Chapter 6.4.3 "List Execution Status", page 107. • release_wait resets the WaitWord that receives the break point number during an interrupt to zero. • If a home jump (defined by home_position or home_position_xyz) has been executed by set_wait, then release_wait leads to a corresponding home return (the INTERNAL-BUSY list execution status is set while the home return is executed). • The wait state can be queried by get_wait_status.
RTC4→RTC6	Basically unchanged functionality. However: Additional PAUSED list execution status . It is set by set_wait and reset with release_wait .
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_wait, get_wait_status, restart_list



Ctrl Command	reset_error
Function	Resets the cumulative error code.
Call	<code>reset_error(Code)</code>
Parameters	<p><code>Code</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.</p>
Comments	<ul style="list-style-type: none"> For error handling see Chapter 6.8 "Error Handling", page 129. The cumulative error code <code>AccError</code> can be reset: <ul style="list-style-type: none"> Bitwise (individually for each error type, for example, Bit #5 and Bit #6 by <code>Code = 2^5 2^6</code> or by <code>Code = RTC6_BUSY RTC6_REJECTED</code>) 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 at get_error. <code>reset_error</code> does not delete the error code of another user program currently assigned access rights to the board. <code>reset_error</code> and <code>n_reset_error</code> are available even without explicit access rights to a specific RTC6 board. The board-specific error variable <code>LastError</code> (see Chapter 6.8 "Error Handling", page 129) is neither generated nor altered by <code>reset_error</code>. In contrast, <code>AccError</code> is altered as specified by <code>Code</code>.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	get_error , get_last_error

Ctrl Command	restart_list
Function	Reenables Signals for “Laser Active” Operation and resumes execution of a list that has been interrupted by pause_list or stop_list .
Call	<code>restart_list()</code>
Comments	<ul style="list-style-type: none"> • restart_list is only executed, if a list has been previously halted by pause_list or stop_list and if the BUSY list execution status and PAUSED list execution status (queryable with get_status) are set. Otherwise (for example, if a list has been halted by set_wait), restart_list is ignored (get_last_error return code RTC6_BUSY). • restart_list resets the PAUSED list execution status. The BUSY list execution status is left unchanged. • In general, an interrupted marking cannot be continued without a disruption in the marking result.
RTC4→RTC6	Basically unchanged functionality. However: Additional PAUSED list execution status that is set with pause_list and stop_list and reset with restart_list .
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	pause_list , stop_list , release_wait

Ctrl Command	rs232_config
Function	Configures the RS-232 interface for the specified baud rate.
Call	<code>rs232_config(BaudRate)</code>
Parameters	<p>BaudRate Baud rate. As an unsigned 32-bit value. Allowed value range: [160 Bd...12.8 MBd].</p>
Comments	<ul style="list-style-type: none"> • See also Chapter 4.6.5 “RS232 Socket Connector”, page 82. • rs232_config is synonymous with uart_config. However, rs232_config has no result. • < DLL 611: value range [300 Bd...+115.200 Bd].
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600. As of DLL 611: extended value range.
References	rs232_write_data , rs232_read_data , uart_config

Ctrl Command	rs232_read_data
Function	Reads a value from the input buffer of the RS-232 interface, see Chapter 9.2.3 "RS-232 Interface", page 287 .
Call	<code>RS232Data = rs232_read_data()</code>
Result	<p>As an unsigned 32-bit value.</p> <p>Bit #0 Next (not yet read by the user program) value of the input buffer. (LSB)</p> <p>...</p> <p>Bit #7</p> <p>Bit #8 "New" bit: = 1: The value is new (has not been previously read). = 0: The value is old (has been already read once by <code>rs232_read_data</code>).</p> <p>Bit #9 0.</p> <p>...</p> <p>Bit #15</p> <p>Bit #16 Number of further (not yet read) characters.</p> <p>...</p> <p>Bit #23</p> <p>Bit #24 Number of buffer overruns.</p> <p>...</p> <p>Bit #31</p>
Comments	<ul style="list-style-type: none"> The RS-232 interface is internally read in internally asynchronously (one character at a time). If this character is new, then it is stored in a 256-character ring buffer. From there, it can be transferred to the user program by <code>rs232_read_data</code> (asynchronously to reading). Byte #0 (Bit #0...Bit #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 (Bit #16...Bit #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 (Bit #24...Bit #31) indicates the number of overruns of the input buffer (a corresponding number of characters were overwritten and therefore irretrievably lost). <p><i>Important: To reset the overflow counter, you must call <code>rs232_read_data</code> correspondingly often!</i></p> <ul style="list-style-type: none"> Example: return value 459098 = 0x0007015A = (0, 7, 1, 90) means: character 90 ('Z') has been read and is new (1), 7 additional characters remain to be read, the buffer has been never overrun (0).

Ctrl Command	rs232_read_data
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	rs232_write_data , uart_config

Ctrl Command	rs232_write_data
Function	Sends a data word (byte) to the RS-232 interface.
Call	<code>rs232_write_data(Data)</code>
Parameters	<p>Data Data word. As an unsigned 32-bit value. Only the least significant byte is transferred to the RS-232 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 "RS-232 Interface", page 286.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	rs232_write_text , rs232_read_data , uart_config , rs232_write_text_list

Ctrl Command	rs232_write_text
Function	Sends a text string (character-by-character) to the RS-232 interface.
Call	<code>rs232_write_text(pData)</code>
Parameters	<p>pData PC memory address of the first character (byte) of the to-be-sent text string. As a pointer to a \0-terminated string.</p>
Comments	<ul style="list-style-type: none"> rs232_write_text is split into an appropriate number of rs232_write_data. During execution of rs232_write_text no further control commands can be executed, but the list execution on the board is not affected. If an executing list itself contains rs232_write_text_list, rs232_write_text should preferably not be used so as to avoid conflicts (race conditions). See also Chapter 9.1.6 "RS-232 Interface", page 286.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	rs232_write_data , rs232_write_text_list , uart_config



Variable List Command	rs232_write_text_list
Function	Sends a text string (character-by-character) to the RS-232 interface.
Call	<code>rs232_write_text_list(pData)</code>
Parameters	<code>pData</code> PC memory address of the first character (byte) of the to-be-sent text string. As a pointer to a \0-terminated string.
Comments	<ul style="list-style-type: none"> When an <code>rs232_write_text_list</code> 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> 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> is similar to that of <code>rs232_write_text</code> (the 12-character block is split into individual characters and sent sequentially to the RS-232 interface). <code>rs232_write_text_list</code> takes some time for execution, depending on the baud rate and text length. See also Chapter 9.1.6 "RS-232 Interface", page 286.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	rs232_write_text , rs232_read_data , uart_config



Ctrl Command	rtc6_count_cards
Function	Returns the number of RTC6 PCIe Boards detected during initialization.
Call	<code>NoOfCards = rtc6_count_cards()</code>
Result	Number of RTC6 PCIe Boards. As an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> Initialization of the installed RTC6 PCIe Boards is to be made separately for each user program by calling init_rtc6_dll. rtc6_count_cards is available even without explicit access rights to a specific RTC6 PCIe Board. rtc6_count_cards 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 129) are neither generated nor altered by rtc6_count_cards.
RTC4→RTC6	New command. Functionality is similar to rtc4_count_cards of the RTC4 command set.
RTC5→RTC6	New command. Functionality is similar to rtc5_count_cards of the RTC5 command set.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	init_rtc6_dll

Delayed Short List Command	save_and_restart_timer
Function	Stores the current value of the RTC6 Timer and resets it to zero.
Call	<code>save_and_restart_timer()</code>
Comments	<ul style="list-style-type: none"> The stored RTC6 Timer value can be read by get_time. save_and_restart_timer is useful for measuring the execution time of a marking process, see Chapter 8.12 "Time Measurements", page 280. The RTC6 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 RTC6-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). A 32-bit "Timestamp Counter" and a 64-bit "Timestamp Counter" are available in addition to the RTC6 Timer, see also Chapter 8.12 "Time Measurements", page 280.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 data medium, ordered by index, and returns the number of thereby stored list commands.
Call	<code>NoOfSavedCommands = save_disk(Name, Mode)</code>
Parameters	<p>Name File name. As a pointer to a \0-terminated ANSI string.</p> <p>Mode Specifies what is to be stored: Bit #0 = 1: All indexed characters and text strings are stored. Bit #1 = 1: All indexed subroutines are stored. Bit #2 Not evaluated. ... Bit #31: Not evaluated. As an unsigned 32-bit value.</p>
Result	The number of list commands saved by save_disk . As an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> • save_disk can be used together with load_disk, for example, to perform defragmentation or to apply subsequent protection to subroutines, see Section "Subsequent Protection and Conversion of Non-Indexed Subroutines", page 115 and Section "Index Management and Defragmentation", page 114. • By save_disk, always the complete sets (no individual characters, text strings or subroutines!) are stored in the specified file. Indexed characters, text strings or subroutines that are referenced multiple times with copy_dst_src are also correspondingly duplicated by save_disk, that is, also stored multiple times. The save_disk command ignores unreferenced non-indexed (not subsequently referenced by set_char_pointer,...) characters, text strings and subroutines. • The number of list commands stored by save_disk can differ from the number of the list commands stored in the protected list memory area "List 3" (= 2^{23} – Mem1 – Mem2 – get_list_space), due to the following: <ul style="list-style-type: none"> – Indexed characters/text strings/subroutines are referenced several times – Indexed characters/text strings/subroutines reside in the unprotected list memory area "List 1" or "List 2" Prior a subsequent load_disk, be sure to compare the returned number with the size of the protected list memory area "List 3" (= 2^{23} – Mem1 – 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 Section "Index Management and Defragmentation", page 114).



Ctrl Command	save_disk
Comments (cont'd)	<ul style="list-style-type: none"> • 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. • save_disk automatically replaces unallowed commands (for example, set_end_of_list) with list_nop commands; missing commands (for example, list_return upon reaching the last memory position of "List 3") are added. • save_disk is not executed (get_last_error return code RTC6_PARAM_ERROR), if: <ul style="list-style-type: none"> – Mode = 0 – Name = NULL • save_disk is not executed (get_last_error return code RTC6_BUSY), if: <ul style="list-style-type: none"> – the BUSY list execution status is set – the INTERNAL-BUSY list execution status is set • save_disk is even executed, if: <ul style="list-style-type: none"> – a list has been paused by set_wait (PAUSED list execution status set) • During the runtime of save_disk: <ul style="list-style-type: none"> – No further commands are executed – External Starts are suppressed • save_disk also stores version information which is checked by load_disk.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 615: version info.
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 .
Call	<code>select_char_set(No)</code>
Parameters	No Number of the desired character set. As an unsigned 32-bit value. Allowed value range: [0...3].
Comments	<ul style="list-style-type: none"> The selection remains valid until another is encountered. The default value (after initialization) is <code>No = 0</code>. If <code>No > 3</code>, then select_char_set is replaced with a list_nop. The current character set then remains valid (get_last_error return code <code>RTC6_PARAM_ERROR</code>). A character set change <i>within</i> a mark_text or mark_text_abs 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→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	mark_text , mark_text_abs

Ctrl Command	select_cor_table
Function	Assigns the previously loaded correction tables to the scan head connector 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 (first scan head connector). = 1...8: Assigns correction table number HeadA to scan head A. As an unsigned 32-bit value. See also number_of_correction_tables.</p> <p>HeadB = 0: Turns off the signals for scan head B (second scan head connector). = 1...8: Assigns correction table number HeadB to scan head B. Requires Option "Second Scan Head Control". As an unsigned 32-bit value. See also number_of_correction_tables.</p>
Comments	<ul style="list-style-type: none"> 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/load_z_table_no (or after a subsequent load_program_file – see below) in order to assign the correction table(s) to the corresponding scan head connector (select_cor_table is automatically called by load_correction_file; see Section "Notes", page 175). select_cor_table and select_cor_table_list issue a jump with jump speed (no “Hard 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 (for example, goto_xy) that sets the scan system’s galvanometer scanners in motion, and select_cor_table or select_cor_table_list should only be started after loading of the desired correction table(s). Otherwise, the galvanometer scanners 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) prior to load_program_file, are not fully effective before select_cor_table or select_cor_table_list is called. select_cor_table is not executed (get_last_error return code RTC6_BUSY), if: <ul style="list-style-type: none"> the BUSY list execution status is set The list command select_cor_table_list can be used without restrictions. select_cor_table is even executed, if: <ul style="list-style-type: none"> a list has been paused by set_wait (PAUSED list execution status set) If the INTERNAL-BUSY list execution status is set, select_cor_table is only executed with a delay (after INTERNAL-BUSY list execution status has been reset again).

Ctrl Command	select_cor_table
Comments (cont'd)	<ul style="list-style-type: none"> The INTERNAL-BUSY list execution status is set while the jump to the corrected galvanometer scanner position is executed. If the values for parameters <code>HeadA</code> or <code>HeadB</code> are invalid (values > <code>number_of_correction_tables</code>) or 0, then <code>select_cor_table</code> turns off the corresponding scan head signals. The galvanometer scanners then remain in their last position. If the Option "Second Scan Head Control" is not enabled, <code>select_cor_table</code> does not assign a correction table to the second scan head connector. The default setting (after <code>load_program_file</code>) is <code>(1, 0)</code>, that is, 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 = 1</code> and <code>HeadB = 0</code>). 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 <code>(0 0)</code>, even if the correction table content is different. If the Option "Second Scan Head Control" is not enabled, 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 first scan head. Even with one scan head, you can thereby quickly switch back and forth between several correction tables, for example, one for a pointer laser and one for the main laser with a different wavelength, see also Chapter 8.5 "Controlling 2D Scan Systems and 3D Scan Systems", page 235. 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 Option "3D" must be enabled. Provided the RTC6's Option "3D" is enabled, you can load several (2D or 3D) correction tables. If the is not enabled, then the xy axis must be connected to the first scan head connector, the z axis to the x or y channel of the second scan head connector. If a 3D correction table is assigned to the first scan head connector, corrected signals for an xy scan head are then transmitted by the first scan head connector and corrected signals for the z axis are transmitted by both channels of the second scan head connector. If multiple RTC6 boards with enabled Option "3D" are installed in a PC, then that many 3-axis systems can be simultaneously controlled.



Ctrl Command	select_cor_table
Comments (cont'd)	<ul style="list-style-type: none"> Provided the Option "3D" and the Option "Second Scan Head Control" are enabled, users specify which signals (xy or z) are to be outputted by which connector when assigning the correction table (see also Chapter 4.5.1 "Scan Head Connectors and Transfer Protocol", page 66 and Section "2D and 3D Correction Files", page 173). 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 (for example, 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 axes and z axis 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 correction table (for example, for <code>select_cor_table(1,1)</code>). Parameters from currently loaded correction tables can be read by get_table_para, or from currently assigned correction tables by get_head_para (see also load_correction_file).
RTC4→RTC6	<p>Unchanged functionality.</p> <p>Exception: up to 8 correction tables in total can be stored in RTC6 memory, see Chapter 8.5.3 "Using Several Correction Tables", page 240. However, two 3D correction tables <i>cannot</i> be simultaneously assigned to the scan head connectors.</p>
RTC5→RTC6	<p>Changed functionality.</p> <ul style="list-style-type: none"> In the RTC6 command set, <code>select_cor_table</code> automatically switches on Zoom and/or Stretch, if a table has been previously loaded by load_stretch_table or load_zoom_correction_file to this table number.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	select_cor_table_list , load_correction_file , load_program_file , load_stretch_table , number_of_correction_tables



Variable List Command	select_cor_table_list
Function	Like select_cor_table , but a list command.
Call	<code>select_cor_table_list(HeadA, HeadB)</code>
Parameters	HeadA Like select_cor_table .
	HeadB Like select_cor_table .
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 scanner position. The extent of this delay depends on the assigned correction table's content for the current galvanometer scanner position (for example, (0 0) after load_program_file); but it is at least 10 μs, even if the correction table does not specify a correction.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	select_cor_table

Ctrl Command	select_rtc
Function	Defines, in a multi-board system, the active RTC6 board for a user program. See Chapter 6.6 "Using Several RTC6 PCIe Boards in One PC", page 122 .
Call	NoOfSelectedCard = select_rtc(CardNo)
Parameters	CardNo RTC6 DLL -internal number of the RTC6 board. As an unsigned 32-bit value.
Result	<p>The returned 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 RTC6 boards found during initialization, no RTC6 Ethernet Board is entered there 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 RTC6_ACCESS_DENIED and possibly RTC6_VERSION_MISMATCH). <p>As an unsigned 32-bit value.</p>
Comments	<ul style="list-style-type: none"> Activation of a board for a user program already occurs when the RTC6 DLL for this user program is initialized (see init_rtc6_dll). The select_rtc command offers the possibility of changing the active board at any time. All user program 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 RTC6_ACCESS_DENIED). If the specified board has no access rights granted and is not acquired by another user program, an attempt is made to acquire it (as by acquire_rtc). 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 RTC6_ACCESS_DENIED RTC6_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 RTC6 boards found during initialization (see rtc6_count_cards), no RTC6 Ethernet Board is entered there or if CardNo = 0 (real boards begin at 1). The return value is then the number of the active board (see above). Therefore, select_rtc(0) can be used to determine the number of the active board at any time. The select_rtc command is available even without explicit access rights to a particular RTC6 board. select_rtc is not available as a multi-board command.



Ctrl Command	select_rtc
RTC4→RTC6	With the RTC4, select_rtc only activates the specified board (unconditionally). With the RTC6, select_rtc additionally delivers a return value and tries to get access to the specified board granted (as with acquire_rtc) or remains ineffective.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	–

Ctrl Command	select_serial_set
Function	Selects a serial-number-set for serial number control commands.
Call	<code>select_serial_set(No)</code>
Parameters	No Number of the desired serial-number-set. As an unsigned 32-bit value. Allowed value 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. select_serial_set does <i>not</i> set the serial-number-set for serial number marking. The list command select_serial_set_list is intended for that purpose. For usage of select_serial_set, see Chapter 7.5.2 "Marking Serial Numbers", page 209.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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>
Parameters	No Number of the desired serial-number-set. As an unsigned 32-bit value. Allowed value 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 usage of select_serial_set_list, see Chapter 7.5.2 "Marking Serial Numbers", page 209.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	select_serial_set , set_serial_step_list , get_list_serial , mark_serial , mark_serial_abs



Ctrl Command	send_user_data
Function	No function.
Comments	<ul style="list-style-type: none">• To date, send_user_data has no effect.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 223 .
Call	<code>set_angle(HeadNo, Angle, 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>first</i> scan head connector. = 2: The definition only affects the <i>second</i> scan head connector. = 0, 3: The definition affects <i>both</i> scan head connectors. = 4: The definition affects the virtual Image field (see also comments). Only the 3 least significant bits are evaluated.</p> <p>Angle Rotation angle. In degrees. 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 Determines when the defined transformation becomes effective. As an unsigned 32-bit value. Like set_matrix.</p>
Comments	<ul style="list-style-type: none"> • See also Chapter 8.2 "Coordinate Transformations", page 223. • As of DLL 617, OUT 617 the following applies for HeadNo = 4: The global coordinate transformations are generally available, even outside a Processing-on-the-fly session.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Last change DLL 617, OUT 617: HeadNo = 4.
References	set_angle_list , set_matrix , set_offset , set_scale



Variable List Command	set_angle_list
Function	Like set_angle , but a list command.
Call	set_angle_list(HeadNo, Angle, at_once)
Parameters	<p>HeadNo Like set_angle. However, HeadNo = 4 is not available.</p> <p>Angle Like set_angle.</p> <p>at_once Determines when the defined transformation becomes effective. As an unsigned 32-bit value.</p> <ul style="list-style-type: none"> = 0: The transformation settings are only accumulated and intermediately stored, but the transformation is not processed as long as this is not activated by another coordinate transformation (for example, 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 accumulated until then) and processed prior to the next list command. If necessary, Signals for "Laser Active" Operation are switched off in advance. = 2: The transformation settings are only accumulated and intermediately stored (as with at_once = 0). However, The transformation is immediately calculated (including all transformation settings that were accumulated and intermediately stored until then) and applied to the current position when the next jump_abs or jump_rel (only if no list is currently being executed: also goto_xy or goto_xyz) is executed. = 3: Like at_once = 1, but the laser control signals remain unaffected. > 3: Like at_once = 2.
Comments	• –
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_angle

Ctrl Command	set_auto_laser_control	
Function	Initializes or deactivates "Automatic Laser Control", see Chapter 7.4.9 ""Automatic Laser Control"" , page 196.	
Call	ErrorCode = set_auto_laser_control(Ctrl, Value, Mode, MinValue, MaxValue)	
Parameters	Ctrl	<p>Control parameter. As an unsigned 32-bit value.</p> <p>= 1...7: Defines which signal parameter is corrected by "Automatic 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 (PulseLength) of the laser control signals LASER1 and LASER2. = 5: Output period (HalfPeriod) of the laser control signals LASER1 and LASER2. = 6: Output value at the 16-bit digital output. = 7: As 5. However, the pulse distance is geometrically constant ("Spot Distance Control"). <p>= 0 or > 7: Deactivates "Automatic Laser Control" (for Ctrl > 7: get_last_error return code RTC6_PARAM_ERROR).</p>
	Value	<p>Defines the 100% value for the parameter selected by Ctrl. As an unsigned 32-bit value.</p> <p>Allowed values:</p> <ul style="list-style-type: none"> For Ctrl = 1/2: 12-bit values [0...4095]. Higher bits are ignored. For Ctrl = 3: 8-bit values [0...255]. Higher bits are ignored. 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...65,535]. Higher bits are ignored.
	Mode	<p>Speed-dependent laser control (Mode = 1...5) lets you select which data is to be used for calculating speed-dependent correction. As an unsigned 32-bit value.</p> <ul style="list-style-type: none"> =1: Set speed. =2: Actual speed (as well as extensions, see below). =3: Reserved. =4: Reserved. =5: Encoder speed (counter pulse rate).

Ctrl Command	set_auto_laser_control															
Parameters (cont'd)	Mode (cont'd)	<p>For <code>Mode</code> = 2 above, the following extensions are available in any combination:</p> <ul style="list-style-type: none"> • <code>M</code> = +4 encoder speed-dependent correction • <code>M</code> = +16 for SCANAhead scan systems (instead of intelliSCAN) • <code>M</code> = +32 The angle-synchronous galvanometer scanner speed is converted into an Image field synchronous speed with the help of the assigned correction table. <p>Then the following applies: <code>Mode</code> = 2 + sum of extensions <code>M</code>.</p> <p>For <code>Mode</code> = 0 or none of the above described ones:</p> <p>Disables the speed-dependent or encoder-speed-dependent laser control (get_last_error-Returncode: <code>RTC6_PARAM_ERROR</code>). The position-dependent laser control always remains effective with a valid value for <code>Ctrl</code>. See also Section "Position-Dependent Laser Control", page 198.</p>														
	MinValue	<p>Defines the <i>lower</i> range limit of the parameter selected by <code>Ctrl</code> for automatic correction and that cannot be undercut.</p> <p>As an unsigned 32-bit value. Allowed values: see <code>Value</code>.</p>														
	MaxValue	<p>Defines the <i>upper</i> range limit of the parameter selected by <code>Ctrl</code> for automatic correction and that cannot be exceeded.</p> <p>As an unsigned 32-bit value. Allowed values: see <code>Value</code>.</p>														
Result	<p><code>ErrorCode</code>.</p> <p>As an unsigned 32-bit value.</p> <table> <tr> <td>0</td><td>No error.</td></tr> <tr> <td>1</td><td>No first scan head active.</td></tr> <tr> <td>2</td><td>No iDRIVE scan system (see Glossary entry on page 26) active.</td></tr> <tr> <td>3</td><td>Invalid <code>Ctrl</code> value.</td></tr> <tr> <td>4</td><td>Invalid <code>Mode</code> value.</td></tr> <tr> <td>5</td><td>Access denied (board locked, get_last_error return code <code>RTC6_ACCESS_DENIED</code>).</td></tr> <tr> <td>6</td><td><code>Mode</code> extension <code>M</code> = +16: No SCANAhead system connected or not active.</td></tr> </table>		0	No error.	1	No first scan head active.	2	No iDRIVE scan system (see Glossary entry on page 26) active.	3	Invalid <code>Ctrl</code> value.	4	Invalid <code>Mode</code> value.	5	Access denied (board locked, get_last_error return code <code>RTC6_ACCESS_DENIED</code>).	6	<code>Mode</code> extension <code>M</code> = +16: No SCANAhead system connected or not active.
0	No error.															
1	No first scan head active.															
2	No iDRIVE scan system (see Glossary entry on page 26) active.															
3	Invalid <code>Ctrl</code> value.															
4	Invalid <code>Mode</code> value.															
5	Access denied (board locked, get_last_error return code <code>RTC6_ACCESS_DENIED</code>).															
6	<code>Mode</code> extension <code>M</code> = +16: No SCANAhead system connected or not active.															
Comments	<ul style="list-style-type: none"> • For usage of set_auto_laser_control, see Chapter 7.4.9 ""Automatic Laser Control"", page 196. • <code>MinValue</code> and <code>MaxValue</code> are automatically exchanged if <code>MinValue</code> > <code>MaxValue</code> and <code>MinValue</code> is clipped to \leq <code>Value</code> and <code>MaxValue</code> clipped to \geq <code>Value</code>. • Control data for speed-dependent laser control is exclusively derived from the scan system data at the first scan head connector and then also applied for the second scan head connector (if that connector has been activated and a scan system is attached). At the time set_auto_laser_control is called, the first scan head connector must already have been assigned a correction table, otherwise error code 1 is returned and <code>Ctrl</code> is set to 0. If only the second scan head connector is being used and/or the scan system at the first scan head connector is shut off, then speed-dependent laser control does not function. 															

Ctrl Command	set_auto_laser_control
Comments (cont'd)	<ul style="list-style-type: none"> For Mode = 2, a functioning iDRIVE scan system must be attached to the first scan head connector. As a result, control_command is unavailable for the first scan head connector as long as this Mode selection is in effect. For Mode = 0 or 1, control_command is available without any restriction. If no functioning iDRIVE scan system is attached, then error code 2 is returned and Ctrl set to 0. Encoder-speed-dependent laser control (Mode = 5) functions even if no scan heads are attached to the scan-head connectors at runtime. The Option Processing-on-the-fly does not need to be activated here either. The target encoder speed is set by set_encoder_speed. With the +4 extension, encoder speeds are converted to galvanometer scanner units (<i>bits/ms</i>) by using the scaling factors from set_fly_x and set_fly_y or set_fly_2d. The result is then added to the current galvanometer scanner speeds. This requires an enabled Option Processing-on-the-fly (in contrast to Mode = 5). The current mark speed is used as reference speed. The set_encoder_speed command (which is used for Mode = 5) is not taken into account with the +4 extension. 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, +4 extension is not effective. set_fly_rot, set_fly_rot_pos, set_fly_x_pos and set_fly_y_pos cannot be combined with the galvanometer scanner speed. For Ctrl = 1...6, the selected signal parameter is updated every 10 μs. For Ctrl = 5 (output period) it is always only effective after an already started laser control signal period has elapsed. For Ctrl = 7, however, the output period is continuously adjusted (from pulse to pulse). Ctrl = 7 ("Spot Distance Control") is only available for excelliSCAN scan heads. For Ctrl = 7 to actually take effect the requested geometric pulse distance must be specified. As long as the geometric pulse distance is 0 (default Value) this control is ineffective, but not deactivated. The actual controlled variable ("temporal pulse distance") cannot be recorded with set_trigger/set_trigger4 and signal 24. The parameters MinValue, MaxValue and Value are ignored. Therefore, the actual controlled variable ("temporal pulse distance") cannot be limited by MinValue and MaxValue. It is automatically controlled to the current mark speed. See also Section "Spot Distance Control", page 201. Activating the angle-to-Image field transformation of the galvanometer scanner speed (+32 extension) is particularly useful, if the correction file has to compensate for strong distortions. If the values for Ctrl or Mode are invalid, then set_auto_laser_control is not executed (return value 3 or 4, get_last_error return code RTC6_PARAM_ERROR). "Automatic Laser Control" should not be combined with the variable laser power of set_multi_mcbsp_in. Only excelliSCAN scan heads: If an excelliSCAN scan head is connected and active, make sure that it is no longer busy before calling set_auto_laser_control (see get_status and state HEAD_BUSY). Otherwise, the not yet processed rest of the marking could be compromised.



Ctrl Command	set_auto_laser_control
RTC4→RTC6	<p>New command.</p> <p>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, the parameter values for Value, MinValue and MaxValue are multiplied by 8. The allowed value ranges decrease accordingly.</p>
RTC5→RTC6	<p>Changed functionality.</p> <p>More laser control modes can be set.</p>
Version info	<p>Available as of DLL 600, OUT 600, RBF 600.</p> <p>Latest changed version DLL 609, OUT 609, RBF 613: Ctrl = 7.</p>
References	load_position_control, load_auto_laser_control, set_auto_laser_params, set_auto_laser_params_list, set_fly_x, set_fly_y, set_fly_2d, set_encoder_speed, spot_distance, spot_distance_ctrl

Ctrl Command	set_auto_laser_params		
Function	Specifies the to-be-controlled signal parameter of the "Automatic Laser Control", the 100% value and its corresponding limit values.		
Call	set_auto_laser_params(<i>Ctrl</i> , <i>Value</i> , <i>MinValue</i> , <i>MaxValue</i>)		
Parameters	<i>Ctrl</i>	The to-be-controlled signal parameter (see set_auto_laser_control). Allowed value range: [1...7].	
	<i>Value</i>	100% value. See set_auto_laser_control .	
	<i>MinValue</i>	Lower range limit. See set_auto_laser_control .	
	<i>MaxValue</i>	Upper range limit. See set_auto_laser_control .	
Comments	<ul style="list-style-type: none"> For information on using set_auto_laser_params, see Chapter 7.4.9 ""Automatic Laser Control"", page 196. The meaning of the parameters is identical to that of 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 <i>Ctrl</i> is invalid (0 or > 7), then set_auto_laser_params is not executed (get_last_error return code RTC6_PARAM_ERROR). 		
RTC4→RTC6	New command. RTC4 Compatibility Mode : see set_auto_laser_control .		
RTC5→RTC6	Unchanged functionality.		
Version info	Available as of DLL 600, OUT 600, RBF 600. Latest changed version DLL 609, OUT 609, RBF 613: <i>Ctrl</i> = 7.		
References	set_auto_laser_control , set_auto_laser_params_list		

Delayed Short List Command	set_auto_laser_params_list		
Function	Like set_auto_laser_params , but a delayed short list command.		
Call	set_auto_laser_params_list(<i>Ctrl</i> , <i>Value</i> , <i>MinValue</i> , <i>MaxValue</i>)		
Parameters	<i>Ctrl</i>	Like set_auto_laser_params .	
	<i>Value</i>	Like set_auto_laser_params .	
	<i>MinValue</i>	Like set_auto_laser_params .	
	<i>MaxValue</i>	Like set_auto_laser_params .	
Comments	<ul style="list-style-type: none"> If the value for <i>Ctrl</i> is invalid (0 or > 7), then set_auto_laser_params_list is replaced with a list_nop (get_last_error return code RTC6_PARAM_ERROR). 		
RTC4→RTC6	New command. RTC4 Compatibility Mode : see set_auto_laser_control .		
RTC5→RTC6	Unchanged functionality.		
Version info	Available as of DLL 600, OUT 600, RBF 600. Latest changed version DLL 609, OUT 609, RBF 613: <i>Ctrl</i> = 7.		
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 value range: [0...1023]. The same applies as for load_char: <code>Char</code> = character set number \times 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 value range: [0...(2²³-1)].</p>
Comments	<ul style="list-style-type: none"> If <code>Char > 1023</code> and/or <code>Pos > (2²³-1)</code>, then set_char_pointer is not executed (get_last_error return code <code>RTC6_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 saving 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 has been finalized by list_return and does not contain set_end_of_list.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 <code>Pos</code> should be entered in the management table. As an unsigned 32-bit value. Allowed value 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 value range: [0...(2²³-1)].</p>
Comments	<ul style="list-style-type: none"> • <code>set_char_table</code> is synonymous with set_text_table_pointer.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_text_table_pointer

Ctrl Command	set_control_mode		
Function	Enables or disables the external control input and resets the counter for External Starts to zero.		
Call	<code>set_control_mode(Mode)</code>		
Parameters	Mode	As an unsigned 32-bit value.	
	Bit #	Value	Description
	Bit #0 (LSB)	= 1:	The external start input (by /START, /START2 or /Slave-START) is enabled.
		= 0:	The external start input is disabled.
	Bit #1	= 1:	With an External Stop (/STOP, /STOP2, /Slave-STOP or simulate_ext_stop) causes explicit cancellation of the external start queue entries (/START, /START2, /Slave-START or simulate_ext_start).
		= 0:	No effect.
	Bit #2	= 1:	The track delay (defined by simulate_ext_start , set_ext_start_delay or set_ext_start_delay_list) that postpones execution of the start relative to the triggering input signal or simulate_ext_start or simulate_ext_start_ctrl command (see Section "External Start", page 289) is deactivated.
		= 0:	No effect. To define and activate the track delay (for example, for Processing-on-the-fly applications), use simulate_ext_start , set_ext_start_delay or set_ext_start_delay_list .
	Bit #3	= 1:	The external start input is <i>not</i> disabled by an external stop signal.
		= 0:	The external start input <i>is</i> disabled by an external stop signal.
	Bit #4		Disables simulate_ext_start_ctrl .
	Bit #5		Reserved.
	Bit #6		Reserved.
	Bit #7		Reserved.
	Bit #8		Reserved.
	Bit #9	= 1:	Encoder resets of the 2 internal encoder counters occur by: <ul style="list-style-type: none"> • An external start signal • simulate_ext_start • simulate_ext_start_ctrl, postponed by a track delay set by simulate_ext_start, set_ext_start_delay or set_ext_start_delay_list, see also Bit #2
		= 0:	Encoder resets occur immediately with each initiating Processing-on-the-fly command.

Ctrl Command	set_control_mode	
Parameters (cont'd)	Bit #10	= 1: 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 Start. The interval between subsequent External Starts (in encoder pulses) is thus constant, see also Section "Regular (Periodic) External Starts", page 292. For <code>stop_execution</code> or an external stop signal, Bit #10 gets reset to "0". = 0: 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 Start has been requested (that is, 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 Starts (in encoder pulses) can thus vary.
	Bit #12	Reserved.

	Bit #31	Reserved.
Comments	<ul style="list-style-type: none"> If execution is aborted by <code>stop_execution</code>, then Bit #0 and Bit #10 gets reset to zero, thus deactivating external start input ports and the counting of track delays with respect to a the 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 execution start. If Bit #9 = 1, then this 10 μs jitter is not present because the encoder reset then occurs synchronously with the start signal. Bit #9 = 0 is useful, when several separate Processing-on-the-fly sessions are to be started within a list. See also Section "External Stop", page 288 and Section "External Start", page 289. 	
RTC4→RTC6	<p>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 RTC6 automatically reserves 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>	
RTC5→RTC6	Unchanged functionality.	
Version info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 605, OUT 605: Bit #4.	
References	<code>set_control_mode_list</code> , <code>set_extstartpos</code> , <code>get_counts</code> , <code>set_max_counts</code> , <code>get_startstop_info</code> , <code>move_to</code>	



Normal List Command	set_control_mode_list
Function	Like set_control_mode , but a list command.
Call	<code>set_control_mode_list(Mode)</code>
Parameters	Mode Like set_control_mode .
Comments	<ul style="list-style-type: none">• The counter for External Starts is <i>not</i> reset by set_control_mode_list.
RTC4→RTC6	Unchanged functionality for the bits that are also used on the RTC4.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 605, OUT 605: 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	<code>PulseLength</code> Default pixel pulse length. As an unsigned 32-bit value. 1 bit equals $1/64 \mu\text{s}$. Allowed value range: $[0 \dots (2^{32}-1)]$.
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 Chapter 8.7.4 "Synchronization", page 267. When initializing (by load_program_file), the <code>PulseLength</code> default value is set to 0.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_port_default

Undelayed Short List Command	set_default_pixel_list
Function	Like set_default_pixel , but a list command.
Call	<code>set_default_pixel_list(PulseLength)</code>
Parameters	<code>PulseLength</code> Like set_default_pixel .
Comments	<ul style="list-style-type: none"> See set_default_pixel.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_default_pixel , set_port_default

Ctrl Command	set_defocus
Function	Determines a focus shift for 3D outputs.
Restriction	If the Option "3D" has not been enabled or if no 3D correction table has been assigned (see select_cor_table), then set_defocus 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>
Parameters	<p>Shift Focus shift. In bits. As a signed 32-bit value. Allowed value range: [-524,288...+524,287]. Out-of-range values are clipped to the boundary values.</p>
Comments	<ul style="list-style-type: none"> A focus shift causes a defocusing of the laser focus relative to the working plane. A positive value increases the focal length of the Dynamic focusing unit and shifts the focus position, for example, for the control value (0 0 0) by about $d = \text{Shift} / K$ to the plane $z = -d$ [mm]. For the calibration factor K, see Chapter 7.3.2 "Image Field Size and Image Field Calibration", page 166; furthermore, 8 in Chapter 7.3.6 "Output Values to the Scan System", page 180. If the resulting total Z output value is too large, the z axis moves to the limit stop. Make sure that this is avoided as far as possible, see get_galvo_controls. If set_defocus 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 output is performed at jump speed. Length and duration of the jump depend on the changed z control value. The duration can be calculated by get_galvo_controls. If no list is currently BUSY list execution status, then the jump executes immediately, whereby no delay occurs at the next start. If the INTERNAL-BUSY list execution status is set, set_defocus is only executed with a delay (after INTERNAL-BUSY list execution status has been reset again). set_defocus sets the INTERNAL-BUSY list execution status while the jump to the changed z output is executed. After "vector-controlled laser control" is activated by set_vector_control(Ctrl = 7), the focus shift changes with parameterized [*]mark[*] Command or Jump commands, too.
RTC4→RTC6	Basically unchanged functionality. But the RTC6 command avoids " Hard jumps ".
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_defocus_list , set_offset_xyz , set_defocus_offset , get_galvo_controls



Variable List Command	set_defocus_list
Function	Like set_defocus , but a list command.
Restriction	Like set_defocus .
Call	<code>set_defocus_list(Shift)</code>
Parameters	Shift Like 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 output. The extent of this delay depends on the size of the specified focus shift; but it is at least 10 μs, even if <code>Shift = 0</code>. • After the jump to the changed z output, a Jump Delay previously configured with set_scanner_delays is inserted. Depending on the dynamics of the z axis it may be reasonable to increase the Jump Delay before calling set_defocus_list. In case of a SCANahead system with automatic delay calculation, long_delay can alternatively be used to compensate for the Tracking error of the z axis.
RTC4→RTC6	See set_defocus .
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_defocus, long_delay



Ctrl Command	set_defocus_offset
Function	Defines an offset in addition to the focus shift (see set_defocus) for all 3D outputs.
Restriction	Like set_defocus .
Call	<code>set_defocus_offset(Shift)</code>
Parameters	Shift Like set_defocus .
Comments	<ul style="list-style-type: none"> • set_defocus_offset has the same effect as set_defocus. • set_defocus_offset enables setting a global defocus shift that is not overwritten by parameterized commands such as para_mark_abs, see set_vector_control(Ctrl = 7). • The set_defocus_offset shift, <ul style="list-style-type: none"> – works additively to set_defocus, – cannot be used as a control parameter for “vector-controlled laser control”, – cannot be recorded via Signal 32 with set_trigger/set_trigger4, – is taken into account by get_z_distance, get_galvo_controls and the back transformation (transform, get_transform).
RTC4→RTC6	New command. In RTC4 Compatibility Mode , the RTC6 multiplies the specified value for Shift by 16.
RTC5→RTC6	Unchanged functionality. In RTC5 Compatibility Mode , the RTC6 multiplies the specified value for Shift by 16.
Version info	Available as of DLL 609, OUT 609, RBF 614.
References	set_defocus , set_defocus_offset_list



Variable List Command	set_defocus_offset_list
Function	Like set_defocus_offset , but a list command.
Restriction	Like set_defocus .
Call	<code>set_defocus_offset_list(Shift)</code>
Parameters	Shift Like set_defocus .
Comments	<ul style="list-style-type: none"> • See set_defocus_offset. • See set_defocus_list. • Even though set_defocus_offset_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→RTC6	New command. In RTC4 Compatibility Mode , the RTC6 multiplies the specified value for Shift by 16.
RTC5→RTC6	Unchanged functionality. In RTC5 Compatibility Mode , the RTC6 multiplies the specified value for Shift by 16.
Version info	Available as of DLL 609, OUT 609, RBF 614.
References	set_defocus_offset , set_defocus_list



Ctrl Command	set_delay_mode		
Function	Turns the “Variable Polygon Delays” mode and the “Variable Jump Delays” 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	Name	Allowed Values	Description
	VarPoly	> 0 = 0	Enables “Variable Polygon Delays” mode. Disables “Variable Polygon Delays” mode. Default setting.
	DirectMove3D	> 0 = 0	As an unsigned 32-bit value. This parameter effects only 3D-applications.
	EdgeLevel	0...(2 ³² -1). 1 bit equals 10 μ s.	The z output is changed directly (linearly) to its end value during a jump. The z output is changed to its end-value in such a way that the focus is kept in one plane during the entire jump. As an unsigned 32-bit value. This parameter defines a maximum “laser on” time for the corners of a Polyline . If the Mark Delay is longer than or equal to this value (because the angle φ is close to 180°, for instance), the laser is switched off (after a LaserOff Delay) and a new Polyline is started. This can be useful for preventing burn-in effects. The EdgeLevel value must be smaller than twice the set value for the Mark Delay , otherwise it has no effect. See also Figure 40 . Note: To disable this feature, the EdgeLevel value must be set to (2 ³² -1) (default value).
	MinJumpDelay	0...(2 ³² -1). 1 bit equals 10 μ s.	As an unsigned 32-bit value. Minimum Jump Delay that cannot be undercut for jumps shorter than JumpLengthLimit (even for jump vectors of zero length, see Figure 36). 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 the Jump Delay . As an unsigned 32-bit value.



Ctrl Command	set_delay_mode
	<p>JumpLengthLimit 0...(2³²-1)</p> <p>Jump length limit. In bits. JumpLengthLimit > 0 enables “Variable Jump Delays” mode. If the jump vector is <i>longer</i> 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” (page 147) between MinJumpDelay and the Jump Delay is calculated and inserted, see Figure 36. JumpLengthLimit = 0 disables “Variable Jump Delays” mode. As an unsigned 32-bit value.</p>
RTC4→RTC6	<p>Unchanged functionality. In addition: extended value range. In RTC4 Compatibility Mode, the RTC6 multiplies the specified value for JumpLengthLimit by 16. The allowed value range decreases accordingly.</p>
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_scanner_delays , load_varpolydelay , set_delay_mode_list

Multiple List Command	set_delay_mode_list
Function	Like set_delay_mode , but a list command.
Call	<code>set_delay_mode_list(VarPoly, DirectMove3D, EdgeLevel, MinJumpDelay, JumpLengthLimit)</code>
Parameters	<p>VarPoly Like set_delay_mode.</p> <p>DirectMove3D Like set_delay_mode.</p> <p>EdgeLevel Like set_delay_mode.</p> <p>MinJumpDelay Like set_delay_mode.</p> <p>JumpLengthLimit Like set_delay_mode.</p>
Comments	<ul style="list-style-type: none"> • See set_delay_mode. • set_delay_mode_list requires two list storage positions. • set_delay_mode_list is executed as two undelayed short list commands. • Any still-pending delayed short list command is executed first.
RTC4→RTC6	<p>New command. RTC4 Compatibility Mode: see set_delay_mode.</p>
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_delay_mode



Ctrl Command	set_dsp_mode
Function	Sets a DSP mode for short-command count.
Call	<code>dsp_mode = set_dsp_mode(Mode)</code>
Parameters	<p>Mode Desired DSP mode. As an unsigned 32-bit value.</p> <ul style="list-style-type: none"> = 0: Corresponds to 500 MHz RTC5 boards (older type series). = 1: Reserved. = 2: Corresponds to 720 MHz RTC5 boards (newer type series). = 3: Corresponds to RTC6 boards. Default setting.
Result	Set DSP mode prior to set_dsp_mode execution. As an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> • set_dsp_mode is used to switch the faster processing of short vectors in standard mode (DSP mode 3) to a RTC5-compatible processing which is slower, see Section "Automatic Delay Adjustments", page 154. • set_dsp_mode is required, if: <ul style="list-style-type: none"> – several RTC6 boards with different DSP modes are to be operated synchronously in a master/slave chain – AND if the adaptation of the short command count is not carried out by sync_slaves • DSP mode > 3 cannot be set. In this case, <code>Mode</code> is clipped to 3.
RTC4→RTC6	New command.
RTC5→RTC6	Basically unchanged functionality. However: Standard mode (DSP mode 3) is available.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	sync_slaves , get_RTC_version



Normal List Command	set_duty_cycle_table
Function	Category: List Commands for a Restricted User Group only. Sets a value in the lookup table for the pulse length.
Call	<code>set_duty_cycle_table(Index, DutyCycle)</code>
Parameters	Index Index in the lookup table for the pulse length. Allowed value range: 0...10. As an unsigned 32-bit value.
	DutyCycle Duty Cycle. In per mille. Allowed value range: 0...1,000. As an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> The relationship between encoder frequency and laser frequency is defined as follows: <ul style="list-style-type: none"> Encoder frequency = $\text{Index} \times [\text{Fmax} / 10]$ Within the lookup table, the values are linearly interpolated. The value for pulse length is set depending on the laser frequency: <ul style="list-style-type: none"> Pulse length = laser frequency $\times \text{DutyCycle} / 1,000$
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 626 , OUT 627 .
References	set_laser_timing_table , regulation3

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 Length of the elliptical half-axis. In bits. As an unsigned 32-bit value. Allowed value range: [1...+8,388,607]. Out-of-range positive values are clipped to the boundary values.</p>
	<p>b See a.</p>
	<p>Phi0 Beginning phase angle (the arc starting point position relative to the end point of half-axis a). In degrees. As a 64-bit IEEE floating point value. A positive sign means "clockwise". Phi0 gets normalized to the value range [0...<360°].</p>
	<p>Phi Arc angle (to-be-marked ellipse section). In degrees. As a 64-bit IEEE floating point value. A positive sign means "clockwise". Allowed value range: [-3,600.0°...+3,600.0°] (± 10 full ellipses). Out-of-range values are clipped to the boundary values.</p>
Comments	<ul style="list-style-type: none"> Specify the to-be-marked elliptical arc's position and orientation in the 2D Image field by mark_ellipse_abs or mark_ellipse_rel. For descriptions of the individual parameters, see Section "Ellipse Commands", page 137. For $a < 1$ and/or $b < 1$, set_ellipse is replaced with a list_nop (get_last_error return code RTC6_PARAM_ERROR).
RTC4→RTC6	<p>New command.</p> <p>In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for a and b by 16. The allowed value range decreases accordingly.</p>
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 <code>Mode</code> = 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 value range: [100.0...16000.0]. Out-of-range values are clipped to the boundary values.</p> <p>Smooth Smoothing factor for a 2-stage low-pass filter. As a 64-bit IEEE floating point value. Allowed value range: [0.0...1.0]. Larger values are clipped.</p>
	<ul style="list-style-type: none"> If <code>Smooth</code> = 0.0, then only the current encoder speed <code>CurrentSpeed</code> (based on counter pulses received in the most recent 10 μs) is used; if <code>Smooth</code> = 1.0, then the speed from the previous clock cycle <code>PreviousSpeed</code>. In general, the used speed is: $\text{Speed} = \text{PreviousSpeed} \times \text{Smooth} + \text{CurrentSpeed} \times (1.0 - \text{Smooth}).$ If <code>Speed</code> \leq 0.0 or <code>Smooth</code> < 0.0, then <code>set_encoder_speed</code> is, already during loading, replaced by a <code>list_nop</code> (<code>get_last_error</code> return code <code>RTC6_PARAM_ERROR</code>). One encoder increment results in 4 counter pulses, see Section "Input Ports for External Encoder Signals", page 298. The maximum value for <code>Speed</code> (16000.0) corresponds to a counting rate of 16 MHz. The minimum value for <code>Speed</code> (100.0) corresponds to a counting rate of 1/(10 μs), that is, 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 Section "Encoder-Speed-Dependent Laser Control", page 203.



Delayed Short List Command	set_encoder_speed
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_encoder_speed_ctrl , set_auto_laser_control

Ctrl Command	set_encoder_speed_ctrl						
Function	Like set_encoder_speed , but a control command.						
Call	<code>set_encoder_speed_ctrl(EncoderNo, Speed, Smooth)</code>						
Parameters	<table> <tr> <td>EncoderNo</td> <td>Like set_encoder_speed.</td> </tr> <tr> <td>Speed</td> <td>Like set_encoder_speed.</td> </tr> <tr> <td>Smooth</td> <td>Like set_encoder_speed.</td> </tr> </table>	EncoderNo	Like set_encoder_speed .	Speed	Like set_encoder_speed .	Smooth	Like set_encoder_speed .
EncoderNo	Like set_encoder_speed .						
Speed	Like set_encoder_speed .						
Smooth	Like set_encoder_speed .						
Comments	<ul style="list-style-type: none"> • set_encoder_speed_ctrl is not executed (get_last_error return code RTC6_PARAM_ERROR), if: <ul style="list-style-type: none"> – Speed ≤ 0.0 – Smooth < 0.0 • set_encoder_speed_ctrl is not executed (get_last_error return code RTC6_BUSY), if: <ul style="list-style-type: none"> – the BUSY list execution status is set • set_encoder_speed_ctrl is even executed, if: <ul style="list-style-type: none"> – a list has been paused by set_wait (PAUSED list execution status set) – the INTERNAL-BUSY list execution status is set 						
RTC4→RTC6	New command.						
RTC5→RTC6	Unchanged functionality.						
Version info	Available as of DLL 600, OUT 600, RBF 600.						
References	set_encoder_speed , set_auto_laser_control						

Normal List Command	set_end_of_list
Function	Ends execution of a list.
Call	<code>set_end_of_list()</code>
Comments	<ul style="list-style-type: none"> If, during processing of a list, the <code>set_end_of_list</code> is encountered and no automatic list change has been previously activated, see Chapter 6.4.6 "Changing Lists Automatically", page 109, then list execution ends. The Signals for "Laser Active" Operation are then switched off and a home jump, if defined by <code>home_position</code> or <code>home_position_xyz</code>, is executed (the INTERNAL-BUSY list execution status is set while the home jump is executed). In contrast, upon reaching a <code>set_end_of_list</code>, execution continues at the other list if an automatic list change has been previously activated. The other list can also be "List 1" if "List 2" has not been configured (<code>Mem2 = 0</code>, see config_list). Upon processing of the <code>set_end_of_list</code>, the USED list status of the respective list (<code>USED1</code> or <code>USED2</code>) is always set and the BUSY list status (<code>BUSY1</code> or <code>BUSY2</code>) of the list is reset, see also Chapter 6.4.3 "List Execution Status", page 107. The BUSY list execution status, on the other hand, is only reset if no automatic list change has been previously activated. An automatic list change of the input pointer never occurs during loading of <code>set_end_of_list</code> (in contrast to an automatic list change of the <i>output pointer</i> during execution of <code>set_end_of_list</code>, if previously activated by <code>auto_change_pos</code> or <code>start_loop</code>). Upon loading <code>set_end_of_list</code>, the READY list status (<code>READY1</code> or <code>READY2</code>) is set and LOAD list status (<code>LOAD1</code> or <code>LOAD2</code>) is reset. Additionally, flushing of the buffered list input is triggered, see Chapter 6.4.1 "Loading Lists", page 104. <code>set_end_of_list</code> is ignored during loading and execution, that is, replaced with a list_nop if an indexed subroutine is currently being loaded or executed (<code>get_last_error</code> return code <code>RTC6_IGNORED</code>).
RTC4→RTC6	Basically unchanged functionality. However: Additional USED list status .
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_start_list



Ctrl Command	set_eth_boot_control
Function	Standalone Functionality : Activates or deactivates automatic booting.
Prerequisite	RTC6 Software Package \geq V1.7.0 and BIOS-ETH \geq 26.
Call	<code>set_eth_boot_control(Ctrl)</code>
Parameters	<p>Ctrl</p> <p>= 0: Deactivates automatic booting. > 0: Activates automatic booting. As an unsigned 32-bit value.</p>
Result	None.
Comments	<ul style="list-style-type: none"> • set_eth_boot_control is not executed (get_last_error return code RTC6_BUSY), if: <ul style="list-style-type: none"> – the BUSY list execution status is set – a list has been paused by set_wait (PAUSED list execution status set) • During the execution of set_eth_boot_control the 10 μs clock cycle of the DSP is interrupted for several ms. • set_eth_boot_control is only allowed with RTC6 Ethernet Boards. Otherwise, a get_last_error return code RTC6_TYPE_REJECTED is generated. • See Chapter 16.7 "Standalone Functionality", page 890. • If set_eth_boot_control has not been successful, Bit 27 is set, see eth_get_last_error and eth_get_error.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 618, OUT 618, RBF 623.
References	eth_boot_dcmd

Ctrl Command	set_extstartpos
Function	Defines the start address (within the range of "List 1" or "List 2") where execution should continue upon External Starts .
Call	<code>set_extstartpos(Pos)</code>
Parameters	<p>Pos Absolute address of the first list command to be executed. As an unsigned 32-bit value. Allowed value range: [0...(2²³-1)].</p>
Comments	<ul style="list-style-type: none"> By default, an External 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 External Starts until a new address is specified by set_extstartpos or set_extstartpos_list. An address range validity check is performed on <code>Pos</code> before each External Start; <code>Pos</code> might therefore become set to 0 at a later point (and remain at this value) if the configuration has been correspondingly changed in the meantime (see config_list). See also Section "External Start", page 289.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_extstartpos_list , set_control_mode

Undelayed Short List Command	set_extstartpos_list
Function	Like set_extstartpos , but a list command.
Call	<code>set_extstartpos_list(Pos)</code>
Parameters	Pos Like set_extstartpos .
Comments	<ul style="list-style-type: none"> set_extstartpos_list can be used within a list, to "link" it to the list that follows. See set_extstartpos.
RTC4→RTC6	Unchanged functionality. In addition: increased value range.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_extstartpos

Ctrl Command	set_ext_start_delay
Function	Sets a track delay for External 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 value 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 Starts, see Section "External Start", page 289. The track delay is specified in <i>relative</i> counting units of the selected encoder counter (the RTC6's encoder counter is triggered by an external or simulated encoder signal, see Chapter 9.3.3 "Synchronization by Encoder Signals", page 297). 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 workpiece's direction of motion and is always positive with simulated encoders). For <code>Delay</code> = 0, the track delay is deactivated. If a track delay is specified, that causes a start trigger (initiated by simulate_ext_start or an external start signal) to occur when the BUSY list execution status or INTERNAL-BUSY list execution status is set (for example, when outputting a list or during goto_xy), then no 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 an External Stop and by stop_execution. They can also be deactivated with set_control_mode/set_control_mode_list (Bit #2). set_ext_start_delay cancels already externally triggered starts that have not yet executed and are still being held in a queue that accommodates up to 8 starts. If <code>EncoderNo</code> > 1, then set_ext_start_delay is ignored (get_last_error return code RTC6_PARAM_ERROR).



Ctrl Command	set_ext_start_delay
RTC4→RTC6	Unchanged functionality. In addition: increased value range. The RTC5 allows this command to be used not only together with an external encoder, but also with an encoder simulation (see Section "Encoder Simulation", page 298) started by simulate_encoder . In RTC4 Compatibility Mode , the RTC5 multiplies the specified value for <code>Delay</code> by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	simulate_ext_start , set_ext_start_delay_list , set_extstartpos , set_extstartpos_list , set_control_mode , set_control_mode_list

Normal List Command	set_ext_start_delay_list
Function	Like set_ext_start_delay , but a list command.
Call	<code>set_ext_start_delay_list(Delay, EncoderNo)</code>
Parameters	<code>Delay</code> Like set_ext_start_delay . <code>EncoderNo</code> Like set_ext_start_delay .
Comments	<ul style="list-style-type: none"> If <code>EncoderNo > 1</code>, then set_ext_start_delay_list is replaced by a list_nop (get_last_error return code <code>RTC6_PARAM_ERROR</code>).
RTC4→RTC6	See set_ext_start_delay .
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_ext_start_delay

Ctrl Command	set_firstpulse_killer
Function	Sets the length of the FirstPulseKiller signal for a YAG laser.
Call	<code>set_firstpulse_killer(Length)</code>
Parameters	<p>Length Length of the FirstPulseKiller signal. As an unsigned 32-bit value. 1 bit equals 1/64 μs. Allowed value range: [0...+(2²⁶-1)]. Out-of-range values are clipped to the boundary values.</p>
Comments	<ul style="list-style-type: none"> The clock frequency connected to the FirstPulseKiller signal is 64 MHz. 1 bit equals 1/64 μs. The signal level is set by set_laser_control. For YAG Mode 2, the Q-Switch delay is also correspondingly altered, see Section "Differences Between the YAG Modes", page 189. In CO₂ Mode, set_firstpulse_killer has no effect. The laser mode is set by set_laser_mode, see Chapter 7.4 "Laser Control", page 183. Q-Switch pulse length and Q-Switch period can be set by set_laser_pulses_ctrl, set_laser_pulses or set_laser_timing.
RTC4→RTC6	Basically unchanged functionality. In addition: increased value range. In RTC4 Compatibility Mode , the RTC6 multiplies the specified value by 8. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_laser_mode , set_laser_timing

Undelayed Short List Command	set_firstpulse_killer_list
Function	Like set_firstpulse_killer , but a list command.
Call	<code>set_firstpulse_killer_list(Length)</code>
Parameters	Length Like set_firstpulse_killer .
RTC4→RTC6	As set_firstpulse_killer .
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_firstpulse_killer



Normal List Command	<code>set_fly_1_axis</code>																
Function	“Fly Extension” Command: Activates a 1 Axis-Processing-on-the-fly application.																
Restriction	If the Option Processing-on-the-fly is not enabled, then <code>set_fly_1_axis</code> terminates the Processing-on-the-fly process (even though it could not have been activated).																
Call	<code>set_fly_1_axis(Axis, Mode, Scale)</code>																
Parameters	<table> <tr> <td>Axis</td><td>Axis from Table 3, page 258. As an unsigned 32-bit value.</td></tr> <tr> <td>Mode</td><td>Mode from Table 4, page 260. As an unsigned 32-bit value.</td></tr> <tr> <td>Scale</td><td>Scaling factor. As a 64-bit IEEE floating point value. Allowed value range: <ul style="list-style-type: none"> • $1/256 \leq \text{Scale} \leq 16.000,0$ with linear axis (1, 2 or 3) • $\text{Scale} > 100,0$ with Rotary axis (4) Scale can be + or -. Only the absolute value is restricted. </td></tr> </table>	Axis	Axis from Table 3, page 258. As an unsigned 32-bit value.	Mode	Mode from Table 4, page 260. As an unsigned 32-bit value.	Scale	Scaling factor. As a 64-bit IEEE floating point value. Allowed value range: <ul style="list-style-type: none"> • $1/256 \leq \text{Scale} \leq 16.000,0$ with linear axis (1, 2 or 3) • $\text{Scale} > 100,0$ with Rotary axis (4) Scale can be + or -. Only the absolute value is restricted.										
Axis	Axis from Table 3, page 258. As an unsigned 32-bit value.																
Mode	Mode from Table 4, page 260. As an unsigned 32-bit value.																
Scale	Scaling factor. As a 64-bit IEEE floating point value. Allowed value range: <ul style="list-style-type: none"> • $1/256 \leq \text{Scale} \leq 16.000,0$ with linear axis (1, 2 or 3) • $\text{Scale} > 100,0$ with Rotary axis (4) Scale can be + or -. Only the absolute value is restricted.																
Comments	<ul style="list-style-type: none"> • Being an “Fly Extension” Command, <code>set_fly_1_axis</code> must not be used mixed with “Classic” Processing-on-the-fly commands (see Footnote, page 241). • See Chapter 8.6 “Processing-on-the-fly”, page 241 and Section ““Fly Extension” Commands”, page 258. • With <code>Mode(1 ... 4)</code>, <code>set_fly_1_axis</code> requires two $10 \mu\text{s}$ clock cycles for execution. • <code>set_fly_1_axis</code> overwrites a previous definition for the same Axis. As a linear axis, <code>set_fly_1_axis</code> can be combined with other linear axes, but not with a Rotary axis. Any still-active rotation-fly application (<code>set_fly_1_axis(Axis =4)</code>) is automatically deactivated and vice versa. It is recommended to explicitly switch off incompatible Processing-on-the-fly corrections beforehand, for example, see <code>fly_return_1_axis</code> and <code>fly_return_2_axes</code>. • With an unallowed parameter value, <code>set_fly_1_axis</code> is replaced by a <code>list_nop</code> (<code>get_last_error</code> return code <code>RTC6_PARAM_ERROR</code>). • The following command calls are executed in the same way: <table> <tr> <td><code>set_fly_1_axis(1, 1, ScaleX)</code></td> <td><code>= set_fly_x(ScaleX)</code></td> </tr> <tr> <td><code>set_fly_1_axis(2, 2, ScaleY)</code></td> <td><code>= set_fly_y(ScaleY)</code></td> </tr> <tr> <td><code>set_fly_1_axis(3, EncoderNo+3, ScaleZ)</code></td> <td><code>= set_fly_z(ScaleZ, EncoderNo)</code></td> </tr> <tr> <td><code>set_fly_1_axis(4, 1, Resolution)</code></td> <td><code>= set_fly_rot(Resolution)</code></td> </tr> <tr> <td><code>set_fly_1_axis(1, 6, ScaleX)</code></td> <td><code>= set_fly_x_pos(ScaleX)</code></td> </tr> <tr> <td><code>set_fly_1_axis(2, 6, ScaleY)</code></td> <td><code>= set_fly_y_pos(ScaleY)</code></td> </tr> <tr> <td><code>set_fly_1_axis(4, 6, Resolution)</code></td> <td><code>= set_fly_rot_pos(Resolution)</code></td> </tr> <tr> <td><code>set_fly_1_axis(1, 7, Scale)</code></td> <td><code>corresponds to set_mcbsp_in(1, Scale)</code></td> </tr> </table> 	<code>set_fly_1_axis(1, 1, ScaleX)</code>	<code>= set_fly_x(ScaleX)</code>	<code>set_fly_1_axis(2, 2, ScaleY)</code>	<code>= set_fly_y(ScaleY)</code>	<code>set_fly_1_axis(3, EncoderNo+3, ScaleZ)</code>	<code>= set_fly_z(ScaleZ, EncoderNo)</code>	<code>set_fly_1_axis(4, 1, Resolution)</code>	<code>= set_fly_rot(Resolution)</code>	<code>set_fly_1_axis(1, 6, ScaleX)</code>	<code>= set_fly_x_pos(ScaleX)</code>	<code>set_fly_1_axis(2, 6, ScaleY)</code>	<code>= set_fly_y_pos(ScaleY)</code>	<code>set_fly_1_axis(4, 6, Resolution)</code>	<code>= set_fly_rot_pos(Resolution)</code>	<code>set_fly_1_axis(1, 7, Scale)</code>	<code>corresponds to set_mcbsp_in(1, Scale)</code>
<code>set_fly_1_axis(1, 1, ScaleX)</code>	<code>= set_fly_x(ScaleX)</code>																
<code>set_fly_1_axis(2, 2, ScaleY)</code>	<code>= set_fly_y(ScaleY)</code>																
<code>set_fly_1_axis(3, EncoderNo+3, ScaleZ)</code>	<code>= set_fly_z(ScaleZ, EncoderNo)</code>																
<code>set_fly_1_axis(4, 1, Resolution)</code>	<code>= set_fly_rot(Resolution)</code>																
<code>set_fly_1_axis(1, 6, ScaleX)</code>	<code>= set_fly_x_pos(ScaleX)</code>																
<code>set_fly_1_axis(2, 6, ScaleY)</code>	<code>= set_fly_y_pos(ScaleY)</code>																
<code>set_fly_1_axis(4, 6, Resolution)</code>	<code>= set_fly_rot_pos(Resolution)</code>																
<code>set_fly_1_axis(1, 7, Scale)</code>	<code>corresponds to set_mcbsp_in(1, Scale)</code>																
RTC4→RTC6	New command.																
RTC5→RTC6	New command.																
Version info	Available as of DLL 617, OUT 617, RBF 623.																
References	<code>set_fly_2_axes</code> , <code>set_fly_3_axes</code>																

Normal List Command	<code>set_fly_2_axes</code>
Function	"Fly Extension" Command: Activates a 2-Axes-Processing-on-the-fly application.
Restriction	If the Option Processing-on-the-fly is not enabled, then <code>set_fly_2_axes</code> terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	<code>set_fly_2_axes(Axis1, Mode1, Scale1, Axis2, Mode2, Scale2)</code>
Parameters	<p>Axis1 Axis from Table 3, page 258. Allowed values: 1...3 (linear axis only). As an unsigned 32-bit value.</p> <p>Mode1 Mode from Table 4, page 260. As an unsigned 32-bit value.</p> <p>Scale1 Scaling factor. As a 64-bit IEEE floating point value. Allowed value range: • $1/256 \leq Scale \leq 16.000,0$ with linear axis (1, 2 or 3) Scale can be + or -. Only the absolute value is restricted.</p> <p>Axis2 Like Axis1.</p> <p>Mode2 Like Mode1.</p> <p>Scale2 Like Scale1.</p>
Comments	<ul style="list-style-type: none"> Being an "Fly Extension" Command, <code>set_fly_2_axes</code> must not be used mixed with "Classic" Processing-on-the-fly commands (see Footnote, page 241). See Chapter 8.6 "Processing-on-the-fly", page 241 and Section "Fly Extension" Commands, page 258. With <code>Mode(1...4)</code>, <code>set_fly_2_axes</code> requires two $10 \mu s$ clock cycles for execution. <code>set_fly_2_axes</code> overwrites a previous definition for the same Axes (even individually). As 2 linear axes, <code>set_fly_2_axes</code> can be combined with a third linear axis, but not with a Rotary axis. Any still-active rotation-fly application (<code>set_fly_1_axis(Axis =4)</code>) is automatically deactivated and vice versa. It is recommended to explicitly switch off incompatible Processing-on-the-fly corrections beforehand, for example, see fly_return_2_axes. Axis1 and Axis2 need to be different linear axes and must not be rotary axes. Mode1 and Mode2 can be the same. If they use the same Encoder, a different scaling factor can be specified by <code>Mode1/Mode2 = 1 / 3 or 2 / 4</code>. With an unallowed parameter value, <code>set_fly_2_axes</code> is replaced by a list_nop (get_last_error return code <code>RTC6_PARAM_ERROR</code>).



Normal List Command	set_fly_2_axes
Comments (cont'd)	<ul style="list-style-type: none"> The following command calls are executed in the same way: <ul style="list-style-type: none"> <code>set_fly_2_axes(1, 1, ScaleX, 2, 2, ScaleY)</code> = <code>set_fly_2d(ScaleX, ScaleY)</code> <code>set_fly_2_axes(1, 1, ScaleX, 2, 2, ScaleY)</code> = <pre> { set_fly_1_axis(1, 1, ScaleX); set_fly_1_axis(2, 2, ScaleY); }</pre> <p>Encoder resets occur:</p> <ul style="list-style-type: none"> With a single <code>set_fly_2_axes</code> call in the same 10 μs clock cycle With two <code>set_fly_1_axis</code> calls in two different 10 μs clock cycles <p><code>set_fly_2_axes(1, 11, Scale, 2, 15, Scale)</code> corresponds to <code>set_mcbsp_in_list(3, Scale)</code></p>
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 617, OUT 617, RBF 623.
References	set_fly_1_axis , set_fly_3_axes

Normal List Command	set_fly_2d				
Function	Activates Processing-on-the-fly correction for compensation of a linear workpiece-movement in 2 dimensions (based on the encoder values transferred to the RTC6 by encoder counters "Encoder0" and "Encoder1"). Sets the corresponding scaling factors.				
Restriction	If the Option Processing-on-the-fly is not enabled, then set_fly_2d terminates the Processing-on-the-fly process (even though it could not have been activated).				
Call	set_fly_2d(ScaleX, ScaleY)				
Parameters	<table> <tr> <td>ScaleX</td> <td>Scaling factor for the x direction (encoder counter "Encoder0"). In bits/count. As a 64-bit IEEE floating point value. Allowed value 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 bits/count. As a 64-bit IEEE floating point value. Allowed value range: $1/256 \leq \text{ScaleY} \leq 16000.0$.</td> </tr> </table>	ScaleX	Scaling factor for the x direction (encoder counter "Encoder0"). In bits/count. As a 64-bit IEEE floating point value. Allowed value range: $1/256 \leq \text{ScaleX} \leq 16000.0$.	ScaleY	Scaling factor for the y direction (encoder counter "Encoder1"). In bits/count. As a 64-bit IEEE floating point value. Allowed value range: $1/256 \leq \text{ScaleY} \leq 16000.0$.
ScaleX	Scaling factor for the x direction (encoder counter "Encoder0"). In bits/count. As a 64-bit IEEE floating point value. Allowed value range: $1/256 \leq \text{ScaleX} \leq 16000.0$.				
ScaleY	Scaling factor for the y direction (encoder counter "Encoder1"). In bits/count. As a 64-bit IEEE floating point value. Allowed value 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 workpiece. The restricted value range applies only to the absolute value. For Processing-on-the-fly correction (for example, determination of the scaling factor or deactivating Processing-on-the-fly correction), see the Chapter 8.6 "Processing-on-the-fly", page 241. For set_fly_2d usage, see the Chapter 8.6.4 "Compensating 2D Motions", page 248. If unallowed parameter values are supplied (for example, for ScaleX = 0), 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 command 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 241. If an encoder compensation has been set by load_fly_2d_table and init_fly_2d, the current encoder values are added to the latest reference values of the 2D encoder compensation and then the sums are saved as new reference values. The encoder counters are then reset to 0. It should <i>not</i> be set by set_control_mode(Bit #9) that the encoder counters are only reset after the subsequent External Start trigger. Otherwise, the reference values of 2D encoder compensation are lost. See also Section "2D Encoder Compensation for xy Positioning Stages", page 248. 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 positioning stage, because here too the reference values are lost. 				



Normal List Command	<code>set_fly_2d</code>
Comments (cont'd)	<ul style="list-style-type: none"> • 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. • You can also use activate_fly_2d/activate_fly_2d_encoder to switch on <code>set_fly_2d</code> Processing-on-the-fly correction.
RTC4→RTC6	<p>New command.</p> <p>In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for <code>ScaleX</code> and <code>ScaleY</code> by 16. The allowed value range decreases accordingly.</p>
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	init_fly_2d , load_fly_2d_table , get_fly_2d_offset , activate_fly_2d , activate_fly_xy

Multiple List Command	set_fly_3_axes
Function	" Fly Extension " Command: Activates a 3-Axes-Processing-on-the-fly application.
Restriction	If the Option Processing-on-the-fly is not enabled, then set_fly_3_axes terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	<code>set_fly_3_axes(ModeX, ScaleX, ModeY, ScaleY, ModeZ, ScaleZ)</code>
Parameters	<p>ModeX Mode from Table 4, page 260. As an unsigned 32-bit value.</p> <p>ScaleX Scaling factor. As a 64-bit IEEE floating point value. Allowed value range: • $1/256 \leq \text{Scale} \leq 16.000,0$ with linear axis (1, 2 or 3) Scale can be + or -. Only the absolute value is restricted.</p> <p>ModeY Like ModeX.</p> <p>ScaleY Like ScaleX.</p> <p>ModeZ Like ModeX.</p> <p>ScaleZ Like ScaleX.</p>
Comments	<ul style="list-style-type: none"> Being an "Fly Extension" Command, set_fly_2_axes must not be used mixed with "Classic" Processing-on-the-fly commands (see Footnote, page 241). See Chapter 8.6 "Processing-on-the-fly", page 241 and Section "Fly Extension" Commands, page 258. With <code>Mode(1...4)</code>, set_fly_3_axes requires two $10 \mu\text{s}$ clock cycles for execution. set_fly_3_axes 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. set_fly_3_axes overwrites a previous definition for the same Axes (even individually). Any still-active rotation-fly application (<code>set_fly_1_axis(Axis =4)</code>) is automatically deactivated and vice versa. It is recommended to explicitly switch off incompatible Processing-on-the-fly corrections beforehand, for example, see fly_return_3_axes. The Axes are automatically set to 1, 2, 3 and therefore, do not need to be specified explicitly. ModeX, ModeY, ModeZ can be same. If they use the same Encoder, a different scaling factor can be specified by ModeX/ModeY/ModeZ = 1 / 3 or 2 / 4. With an unallowed parameter value, set_fly_3_axes is replaced by a list_nop (get_last_error return code <code>RTC6_PARAM_ERROR</code>).



Multiple List Command	set_fly_3_axes
Comments (cont'd)	<ul style="list-style-type: none"> The following command calls are executed in the same way: <ul style="list-style-type: none"> - <code>set_fly_3_axes(1, ScaleX, 2, ScaleY, 3 or 4, ScaleZ) =</code> <pre> { set_fly_1_axis(1, 1, Scale X); set_fly_1_axis(2, 2, Scale Y); set_fly_1_axis(3, 3 or 4, Scale Z); }</pre> - <code>set_fly_3_axes(1, ScaleX, 2, ScaleY, 3 or 4, ScaleZ) =</code> <pre> { set_fly_2_axes(1, 1, Scale1 X, 2, 2, Scale2 X); set_fly_1_axis(3, 3 or 4, Scale Z); }</pre> <p>Encoder resets occur:</p> <ul style="list-style-type: none"> With a single <code>set_fly_3_axes</code> call in the same 10 μs clock cycle With three <code>set_fly_1_axis</code> calls in three different 10 μs clock cycles With one <code>set_fly_2_axes</code> call and one <code>set_fly_1_axis</code> call in two different 10 μs clock cycles <p>- <code>set_fly_3_axes(8, 1, 0, 12, 1, 0, 16, 1, 0)</code> corresponds to <code>set_multi_mcbsp_in_list(Ctrl, P, Mode)</code></p>
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 617, OUT 617, RBF 623.
References	set_fly_1_axis , set_fly_2_axes



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	<p>Xmin Range boundary. As a signed 32-bit value. Allowed value range: [-524,288...+524,287]. Out-of-range values are clipped to the boundary values.</p> <p>Xmax Like Xmin (analogously).</p> <p>Ymin Like Xmin (analogously).</p> <p>Ymax Like Xmin (analogously).</p>
Comments	<ul style="list-style-type: none"> For usage of set_fly_limits, see Section "Customer-Defined Monitoring Area", page 255. During initialization (with load_program_file), the boundary limits get set to the following default values: <ul style="list-style-type: none"> - Xmin = Ymin = -524,288 - Xmax = Ymax = +524,287 Range boundaries specified using the parameter value 0 are set to the above-mentioned default values.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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><code>Zmin</code> Range boundary. As a signed 32-bit value. Allowed value range: [-524,288...+524,287]. Out-of-range values are clipped to the boundary values.</p> <p><code>Zmax</code> Like <code>Zmin</code> (analogously).</p>
Comments	<ul style="list-style-type: none"> For usage of <code>set_fly_limits_z</code>, see also Chapter 8.6.9 "Monitoring Processing-on-the-fly Corrections", page 254. During initialization (with load_program_file), the boundary limits get set to the following default values: <ul style="list-style-type: none"> <code>Zmin</code> = -524,288 <code>Zmax</code> = +524,287 Boundary limits specified using the parameter value 0 also get set to the above-mentioned default values.
RTC4→RTC6	New command. In RTC4 Compatibility Mode , the RTC6 multiplies the specified values for <code>Zmin</code> and <code>Zmax</code> by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality. In RTC5 Compatibility Mode , the RTC6 multiplies the specified values for <code>Zmin</code> and <code>Zmax</code> by 16. The allowed value range decreases accordingly.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 RTC6 by encoder counter "Encoder0") and sets the corresponding <code>Resolution</code> parameter.
Restriction	If the Option Processing-on-the-fly is not enabled, then <code>set_fly_rot</code> terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	<code>set_fly_rot(Resolution)</code>
Parameters	<code>Resolution</code> Number of steps per revolution. As a 64-bit IEEE floating point value. Allowed value 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 8.6 "Processing-on-the-fly", page 241. Before executing <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 (for example, 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 command or "Arc" command (without "set_fly_rot" Processing-on-the-fly correction). The various Processing-on-the-fly corrections cannot be arbitrarily combined, see Section "Overview", page 241. For deactivating Processing-on-the-fly correction, see Chapter 8.6.5 "Deactivating Processing-on-the-fly Correction", page 250. By <code>set_control_mode(Bit #9)</code>, it can be set in advance when the encoder counter "Encoder0" is reset by <code>set_fly_rot</code>.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_rot_center , set_rot_center_list , fly_return , get_encoder , set_fly_rot_pos

Normal List Command	<code>set_fly_rot_pos</code>
Function	Activates Processing-on-the-fly correction for compensation of workpiece or scan system rotary movement (based on angular position values transferred to the RTC6 by the McBSP interface). It thereby sets the corresponding <code>Resolution</code> parameter.
Restriction	If the Option Processing-on-the-fly is not enabled, then <code>set_fly_rot_pos</code> terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	<code>set_fly_rot_pos(Resolution)</code>
Parameters	<code>Resolution</code> Number of steps per revolution. As a 64-bit IEEE floating point value. Allowed value 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 8.6 "Processing-on-the-fly", page 241. Before executing <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 used. If unallowed parameter values are supplied (for example, 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 command 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 Section "Overview", page 241. For deactivating Processing-on-the-fly correction, see Chapter 8.6.5 "Deactivating Processing-on-the-fly Correction", page 250. The McBSP interface cannot be simultaneously used for both Processing-on-the-fly applications and Online Positioning. See also Section "Notes", page 229. The McBSP interface ignores the first FrameSync signal after a <code>load_program_file</code> or <code>mcbsp_init</code>. That is, data provided is not transmitted, see page 85.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_rot_center , set_rot_center_list , fly_return , read_mcbsp , set_fly_rot



Ctrl Command	set_fly_tracking_error
Function	No function.
Call	<code>set_fly_tracking_error(TrackingErrorX, TrackingErrorY)</code>
Parameters	<code>TrackingErrorX</code> Tracking error for the x axis. In units of 10 μs. As a signed 32-bit value. Allowed value range: [0...65,535]. Excessive bits are ignored. <code>TrackingErrorY</code> Tracking error for the y axis. Otherwise, like <code>TrackingErrorX</code> .
Comments	<ul style="list-style-type: none">• To date, this command has no effect.
RTC4→RTC6	New command.
RTC5→RTC6	Changed functionality. To date, this command has no effect.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	–

Normal List Command	<code>set_fly_x</code>
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 RTC6 by encoder counter "Encoder0") and sets the corresponding scaling factor.
Restriction	If the Option Processing-on-the-fly is not enabled, then <code>set_fly_x</code> terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	<code>set_fly_x(ScaleX)</code>
Parameters	<p>ScaleX Scaling factor for the x direction (encoder counter "Encoder0"). In bits/count. As a 64-bit IEEE floating point value. Allowed value range: $1/256 \leq \text{ScaleX} \leq 16000.0$.</p>
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 8.6 "Processing-on-the-fly", page 241. If unallowed parameter values are supplied (for example, for ScaleX = 0), 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 command or "Arc" command (without "set_fly_x" Processing-on-the-fly correction). The various Processing-on-the-fly corrections cannot be arbitrarily combined, see Section "Overview", page 241. For deactivating Processing-on-the-fly correction, see Chapter 8.6.5 "Deactivating Processing-on-the-fly Correction", page 250. By <code>set_control_mode(Bit #9)</code>, it can be set in advance when the encoder counter "Encoder0" is reset by <code>set_fly_x</code>. You can also switch on <code>set_fly_x/set_fly_y</code> Processing-on-the-fly correction by activate_fly_xy/activate_fly_xy_encoder.
RTC4→RTC6	Unchanged functionality. In addition: changed value range. In RTC4 Compatibility Mode , the RTC6 multiplies the specified value for ScaleX by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 RTC6 by the McBSP interface); thereby sets the corresponding scaling factor.
Restriction	If the Option Processing-on-the-fly is not enabled, then set_fly_x_pos terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	<code>set_fly_x_pos(ScaleX)</code>
Parameters	ScaleX Scaling factor for the x direction in <i>(RTC6)bits/(McBSP)bit</i> . As a 64-bit IEEE floating point value. Allowed value range: $1/256 \leq \text{ScaleX} \leq 16000.0$.
Comments	<ul style="list-style-type: none"> • <code>ScaleX</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 8.6 "Processing-on-the-fly", page 241. • If unallowed parameter values are supplied (for example, for <code>ScaleX = 0</code>), then set_fly_x_pos does not activate a Processing-on-the-fly correction or deactivates a Processing-on-the-fly correction previously activated by set_fly_x_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 command or "Arc" command (without "set_fly_x_pos" Processing-on-the-fly correction). • The various Processing-on-the-fly corrections cannot be arbitrarily combined, see Section "Overview", page 241. • For deactivating Processing-on-the-fly correction, see Chapter 8.6.5 "Deactivating Processing-on-the-fly Correction", page 250. • For 1D correction (when only set_fly_x_pos is used), the McBSP interface provides a (signed) 32-bit value. In contrast, only a (signed) 16-bit value per axis is supplied for 2D correction (set_fly_x_pos and set_fly_y_pos). Here, set_fly_x_pos uses the McBSP 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 interface cannot be simultaneously used for both Processing-on-the-fly applications and Online Positioning. See also Section "Notes", page 229. • The McBSP interface ignores the first FrameSync signal after a load_program_file or mcbsp_init. That is, data provided is not transmitted, see page 85.
RTC4→RTC6	New command. In RTC4 Compatibility Mode , the RTC6 multiplies the specified value for <code>ScaleX</code> by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	fly_return , set_fly_y_pos , read_mcbsp , set_fly_x , set_fly_y

Normal List Command	<code>set_fly_y</code>
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 RTC6 by encoder counter "Encoder1") and sets the corresponding scaling factor.
Restriction	If the Option Processing-on-the-fly is not enabled, then <code>set_fly_y</code> terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	<code>set_fly_y(ScaleY)</code>
Parameters	<code>ScaleY</code> Scaling factor for the y direction (encoder counter "Encoder1"). In bits/count. As a 64-bit IEEE floating point value. Allowed value range: $1/256 \leq \text{ScaleY} \leq 16000.0$.
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 8.6 "Processing-on-the-fly", page 241. If unallowed parameter values are supplied (for example, 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 command or "Arc" command (without "set_fly_y" Processing-on-the-fly correction). The various Processing-on-the-fly corrections cannot be arbitrarily combined, see Section "Overview", page 241. For deactivating Processing-on-the-fly correction, see Chapter 8.6.5 "Deactivating Processing-on-the-fly Correction", page 250. By <code>set_control_mode(Bit #9)</code>, it can be set in advance when the encoder counter "Encoder0" is reset by <code>set_fly_y</code>. You can also switch on <code>set_fly_x</code>/<code>set_fly_y</code> Processing-on-the-fly correction by activate_fly_xy/activate_fly_xy_encoder.
RTC4→RTC6	Unchanged functionality. In addition: changed value range. In RTC4 Compatibility Mode , the RTC6 multiplies the specified value for <code>ScaleY</code> by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 RTC6 by the McBSP interface); thereby sets the corresponding scaling factor.
Restriction	If the Option Processing-on-the-fly is not enabled, then set_fly_y_pos terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	<code>set_fly_y_pos(ScaleY)</code>
Parameters	ScaleY Scaling factor for the y direction in (RTC6) <i>bits</i> /(McBSP) <i>bit</i> . As a 64-bit IEEE floating point value. Allowed value 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 8.6 "Processing-on-the-fly", page 241. If unallowed parameter values are supplied (for example, 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 command 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 Section "Overview", page 241. For deactivating Processing-on-the-fly correction, see Chapter 8.6.5 "Deactivating Processing-on-the-fly Correction", page 250. For 1D correction (when only set_fly_y_pos is used), the McBSP interface provides a (signed) 32-bit value. In contrast, only a (signed) 16-bit value per axis is supplied for 2D correction (set_fly_x_pos and set_fly_y_pos). Here, set_fly_x_pos uses the McBSP 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 interface cannot be simultaneously used for both Processing-on-the-fly applications and Online Positioning. See also Section "Notes", page 229. The McBSP interface ignores the first FrameSync signal after a load_program_file or mcbsp_init. That is, data provided is not transmitted, see page 85.
RTC4→RTC6	New command. In RTC4 Compatibility Mode , the RTC6 multiplies the specified value for ScaleY by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 RTC6 by the specified encoder counter) and sets the corresponding scaling factor.				
Restriction	If the Option Processing-on-the-fly is not enabled, then <code>set_fly_z</code> terminates the Processing-on-the-fly process (even though it could not have been activated).				
Call	<code>set_fly_z(ScaleZ, EncoderNo)</code>				
Parameters	<table> <tr> <td>ScaleZ</td> <td>Scaling factor for the z direction. In bits/count. As a 64-bit IEEE floating point value. Allowed value range: $1/256 \leq \text{ScaleZ} \leq 16,000.0$.</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>	ScaleZ	Scaling factor for the z direction. In bits/count. As a 64-bit IEEE floating point value. Allowed value range: $1/256 \leq \text{ScaleZ} \leq 16,000.0$.	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".
ScaleZ	Scaling factor for the z direction. In bits/count. As a 64-bit IEEE floating point value. Allowed value range: $1/256 \leq \text{ScaleZ} \leq 16,000.0$.				
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"> <code>ScaleZ</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 8.6 "Processing-on-the-fly", page 241. If unallowed <code>ScaleZ</code> parameter values are supplied (for example, for <code>ScaleZ = 0</code>), then <code>set_fly_z</code> does not activate a Processing-on-the-fly correction or deactivates a Processing-on-the-fly correction previously activated by <code>set_fly_z</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 command 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 257. By <code>set_control_mode(Bit #9)</code>, it can be set in advance when the encoder counter "Encoder0" or "Encoder1" is reset by <code>set_fly_z</code>. If <code>EncoderNo > 1</code>, <code>set_fly_z</code> is replaced by a <code>list_nop (get_last_error</code> return code <code>RTC6_PARAM_ERROR</code>). 				
RTC4→RTC6	New command. In RTC4 Compatibility Mode , the RTC6 multiplies the specified value for <code>ScaleZ</code> by 16. The allowed value range decreases accordingly.				
RTC5→RTC6	Unchanged functionality.				
Version info	Available as of DLL 600, OUT 600, RBF 600.				
References	fly_return_z				

Ctrl Command	set_free_variable
Function	Sets a free variable to the desired value.
Call	<code>set_free_variable(No, Value)</code>
Parameters	No Number of the free variable to be set. As an unsigned 32-bit value. Allowed value range: [0...7]. Only the 3 least significant bits are evaluated.
	Value Desired variable value. As an unsigned 32-bit value. Allowed value range: [0...(2 ³² -1)].
Comments	<ul style="list-style-type: none"> See Chapter 6.9.1 "Free Variables", page 134. Standalone Functionality: <code>set_free_variable</code> is a control command allowed for automatic booting (although the corresponding list command <code>set_free_variable_list</code> exists), see Table 8, page 895 in Chapter 16.7 "Standalone Functionality", page 890. By <code>set_free_variable</code>, any version code can be assigned to the Boot Image, which in PC operation can be read and checked before an /START.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 618: control command allowed for automatic booting, see page 894 .
References	set_free_variable_list , get_free_variable , set_trigger , eth_boot_dcmsg

Undelayed Short List Command	set_free_variable_list
Function	Like <code>set_free_variable</code> , but a list command.
Call	<code>set_free_variable_list(No, Value)</code>
Parameters	No Like <code>set_free_variable</code> .
	Value Like <code>set_free_variable</code> .
Comments	<ul style="list-style-type: none"> See set_free_variable.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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	HeadNo Number of the scan head connector. As an unsigned 32-bit value. Allowed values: = 1: First scan head connector. = 2: Second scan head connector.
	GalvoGainX x gain value. As a 64-bit IEEE floating point value. Allowed value range: [0.01...100].
	GalvoGainY Like GalvoGainX (analogously).
	GalvoOffsetX x offset value. In bits. As a signed 32-bit value. Allowed value range: [-524,288...+524,287].
	GalvoOffsetY Like GalvoOffsetX (analogously).
Comments	<ul style="list-style-type: none"> For usage of <code>set_hi</code>, see Section "Customer-Specific Calibration", page 277. The specified gain and offset values overwrite the values that were set by <code>auto_cal</code> and can themselves be overwritten by a subsequent call of <code>auto_cal</code>. With changed gain and offset values, the transition is automatically performed at the predefined jump speed (see <code>set_jump_speed</code>). <code>set_hi</code> is not executed, if: <ul style="list-style-type: none"> A parameter value is invalid (<code>get_last_error</code> return code <code>RTC6_PARAM_ERROR</code>) the BUSY list execution status is set (<code>get_last_error</code> return code <code>RTC6_BUSY</code>) the INTERNAL-BUSY list execution status is set (<code>get_last_error</code> return code <code>RTC6_BUSY</code>) <code>set_hi</code> is even executed, if: <ul style="list-style-type: none"> a list has been paused by <code>set_wait</code> (PAUSED list execution status set) For <code>RTC6_PARAM_ERROR</code>, the BUSY list execution status is not checked. Therefore the return codes <code>RTC6_BUSY</code> and <code>RTC6_PARAM_ERROR</code> do not occur simultaneously. If the Option "Second Scan Head Control" has not been enabled, values specified for the second scan head control have no effect.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	auto_cal , get_hi_pos , write_hi_pos



Ctrl Command	set_input_pointer
Function	Opens the list memory for writing with list commands and sets the input pointer to the specified <i>absolute</i> address in list memory ("List 1" or "List 2").
Call	<code>set_input_pointer(Pos)</code>
Parameters	<p>Pos Position (<i>absolute</i> memory address) of the input pointer. $[0\dots(2^{23}-1)]$. As an unsigned 32-bit value.</p>
Comments	<ul style="list-style-type: none"> The next list command is stored at the specified address and all further list commands at the subsequent addresses in the selected list. <code>set_input_pointer</code> performs basically like <code>set_start_list_pos</code> (see comments there). But <code>set_input_pointer</code> sets the input pointer based on a specified <i>absolute</i> memory address, whereas <code>set_start_list_pos</code> uses a specified list number and a <i>relative</i> memory address. For $\text{Pos} \geq \text{Mem1} + \text{Mem2}$ (see <code>config_list</code>), <code>Pos</code> is set to 0.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 <code>MaskSet</code> , if the current <code>IOvalue</code> at the 16-bit digital input port on the EXTENSION 1 socket connector meets the following condition: $((\text{IOvalue} \text{ AND } \text{Mask1}) = \text{Mask1}) \text{ AND } (((\text{not IOvalue}) \text{ AND } \text{Mask0}) = \text{Mask0})$ (= if the bits specified in <code>Mask1</code> are 1 and the bits specified in <code>Mask0</code> are 0).
Call	<code>set_io_cond_list(Mask1, Mask0, MaskSet)</code>
Parameters	<p><code>Mask1</code> 16-bit mask. As an unsigned 32-bit value. Only the lower 16 bits are evaluated.</p> <p><code>Mask0</code> Like <code>Mask1</code>.</p> <p><code>MaskSet</code> Like <code>Mask1</code>.</p>
Comments	<ul style="list-style-type: none"> • <code>set_io_cond_list</code> sets only those bits of the digital output port that are set in the parameter <code>MaskSet</code> and leaves the other bits unchanged. • See Section "16-Bit Digital Input Port and 16-Bit Digital Output Port", page 77 and Chapter 9.3.2 "Conditional Command Execution", page 294.
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 Bit #1...Bit #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→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	clear_io_cond_list , write_io_port , write_io_port_mask , get_io_status , read_io_port

Ctrl Command	set_jump_mode
Function	Enables and activates or disables and deactivates Jump Mode for 2D jumps and sets the related parameters.
Prerequisite	Enabling is only possible if a jump-tuning-equipped intelliSCAN (with scan system firmware version \geq 2078) is attached to at least one of the two scan head connectors. Otherwise, set_jump_mode has no effect.
Call	<code>ErrorCode = set_jump_mode(Flag, Length, VA1, VA2, VB1, VB2, JA1, JA2, JB1, JB2)</code>
Parameters	<p>Flag Switching flag. As a signed 32-bit value. Allowed values: = -1: Jump Mode is disabled. Afterward, switching the Flag by set_jump_mode_list is <i>no longer</i> possible. = 0: Jump Mode is enabled but deactivated. Afterwards it is also possible to switch the flag by set_jump_mode_list. = 1: Jump Mode is enabled and activated. Afterwards it is also possible to switch the flag by set_jump_mode_list.</p> <p>Length Limit of the jump length (per axis) under which 2D jumps – even with activated Jump Mode – are performed in vector mode. As an unsigned 32-bit value.</p> <p>VA1 Numbers of the tunings that should be used for jump execution in Jump Mode: VA2 V: The Vector tuning that is set at the end of a 2D jump. VB1 J: The Jump tuning that is set at the beginning of a 2D jump. VB2 A First scan head connector. JA1 B Second scan head connector. JA2 1: x axis (STATUS channel, Galvanometer scanner 2). 2: y axis (STATUS1 channel, Galvanometer scanner 1). JB1 Allowed values: JB2 = -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. As a signed 32-bit value.</p>

Ctrl Command	set_jump_mode
Result	<p>Error code. As a signed 32-bit value.</p> <p>0 No error: Flag successfully switched to 0 (Jump Mode deactivated, vector mode activated).</p> <p>1 No error: Flag successfully switched to 1 (Jump Mode activated, vector mode deactivated).</p> <p>-1 Flag successfully switched to -1 (Jump Mode deactivated and disabled).</p> <p>-2 Busy error: board BUSY list execution status or INTERNAL-BUSY list execution status (get_last_error return code RTC6_BUSY).</p> <p>-3 Board not responding: no program loaded or PCI error (get_last_error return code RTC6_TIMEOUT).</p> <p>-4 Access error: board reserved for another user program (get_last_error return code RTC6_ACCESS_DENIED).</p> <p>> 1 Did not pass the check (see also notes). Flag is set to -1 (Jump Mode deactivated and disabled).</p> <p>The following is returned:</p> <p>Byte #0 = 255. Byte #1 = Error code for first scan head connector. Byte #2 = Error code for second scan head connector. Byte #3 = 0.</p> <p>Whereby error code:</p> <ul style="list-style-type: none"> =1: x axis (galvanometer scanner 2) not responding or no intelliSCAN (with scan system firmware version \geq 2078) attached. =2: y axis (galvanometer scanner 1) not responding or no intelliSCAN (with scan system firmware version \geq 2078) attached. =4: no correction table assigned. =8: incorrect tuning number(s): incorrect type or unsuitable for rapid switching.

Ctrl Command	set_jump_mode
Comments	<ul style="list-style-type: none"> For usage of <code>set_jump_mode</code>, see Chapter 8.1.5 "Jump Mode", page 216. A check (see also Section "Requirements and Activation", page 217) is only performed if <code>Flag</code> = -1 (the initialization state) prior to the <code>set_jump_mode</code> call and/or if the supplied tuning numbers do not match those stored on the board. Otherwise, only the flag is switched. <p>For the check, the board must not be BUSY list execution status or INTERNAL-BUSY list execution status, because meanwhile the data type to-be-returned by the scan system 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, <code>Flag</code> 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 <code>Flag</code> = 1 (Jump Mode activated, vector mode deactivated) and <code>Flag</code> = 0 (Jump Mode deactivated, vector mode activated) without another check having to be performed.</p> <ul style="list-style-type: none"> Use -1 as the tuning number if certain tunings should not be checked (for example, 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 Option "Second Scan Head Control" is not enabled, then the tuning numbers for the second scan head connector (B) is automatically set to -1 (even if others were supplied). If all tuning numbers are -1, then <code>Flag</code> is set to -1 (return value -1). Even after successful activation of Jump Mode (<code>Flag</code> = 1), the first servo switching only occurs after the first subsequent 2D jump (see Section "Functional Principle", page 216). If the currently set tuning then does not match the Jump tuning 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 > 1, error code = 2), 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 has been supplied by <code>set_jump_mode</code>.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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.
Prerequisite	See set_jump_mode .
Call	<code>set_jump_mode_list(Flag)</code>
Parameters	Flag See set_jump_mode .
Comments	<ul style="list-style-type: none"> For usage of set_jump_mode_list, see Chapter 8.1.5 "Jump Mode", page 216. set_jump_mode_list functions like the control command set_jump_mode (see notes there) but, as a list command, has the following differences: <ul style="list-style-type: none"> No tunings or jump length limit can be specified by set_jump_mode_list. set_jump_mode_list does not perform a check. <code>Flag</code> must have previously been successfully set to 0 or 1 by set_jump_mode. Though Jump Mode can be deactivated and disabled by setting <code>Flag</code> to -1 by set_jump_mode or set_jump_mode_list, it can only be reactivated again by set_jump_mode (if the check is successful). If <code>Flag</code> has been -1 prior to calling set_jump_mode_list, then set_jump_mode_list has no effect.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
Reference	set_jump_mode

Undelayed Short List Command	set_jump_speed
Function	Defines the jump speed for Vector commands .
Call	<code>set_jump_speed(Speed)</code>
Parameters	Speed Jump speed. In Bits/ms. As a 64-bit IEEE floating point value. Allowed value range: [1.6...800000.0].
Comments	<ul style="list-style-type: none"> By default a jump speed of 10000 <i>bits/ms</i> is preset. The specified jump speed is used for all Jump commands until a new value is specified. The actual jump speed v_{jump} in the image plane in m/s is derived from the specified Speed value [<i>bits/ms</i>] and the calibration factor K [Bits/mm] as follows: $v_{jump} = \text{Speed} / K$ <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"> set_jump_speed is also available as control command set_jump_speed_ctrl.
RTC4→RTC6	Unchanged functionality. In RTC4 Compatibility Mode , the RTC6 multiplies the specified Speed value by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	jump_abs, jump_rel, set_mark_speed, set_jump_speed_ctrl

Ctrl Command	set_jump_speed_ctrl
Function	Like set_jump_speed , but a control command.
Call	<code>set_jump_speed_ctrl(Speed)</code>
Parameters	Speed Like set_jump_speed .
Comments	<ul style="list-style-type: none"> set_jump_speed_ctrl is not executed (get_last_error return code RTC6_BUSY), if: <ul style="list-style-type: none"> the BUSY list execution status is set set_jump_speed_ctrl is even executed, if: <ul style="list-style-type: none"> a list has been paused by set_wait (PAUSED list execution status set) the INTERNAL-BUSY list execution status is set
RTC4→RTC6	New command. In RTC4 Compatibility Mode : like set_jump_speed .
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_jump_speed, set_mark_speed, set_mark_speed_ctrl



Ctrl 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	<p>Error code. As an unsigned 32-bit value.</p> <p>0 No error.</p> <p>1 Busy error: board BUSY list execution status or INTERNAL-BUSY list execution status (get_last_error return code RTC6_BUSY).</p> <p>4 Verify error: DSP memory error.</p> <p>11 RTC6 board driver error.</p>
Comments	<ul style="list-style-type: none"> • set_jump_table is not executed (get_last_error return code RTC6_BUSY), if: <ul style="list-style-type: none"> – the BUSY list execution status is set – the INTERNAL-BUSY list execution status is set • set_jump_table is even executed, if: <ul style="list-style-type: none"> – a list has been paused by set_wait (PAUSED list execution status set)
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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>
Parameters	<p>Ctrl As an unsigned 32-bit value.</p> <p>Bit #0 Pulse Switch Setting (does not concern Laser Mode 4 or Laser Mode 6). (LSB) The setting only affects 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 52 and Figure 53. = 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. See Pulse Completion, page 185.</p> <p>Bit #1 Phase shift of the laser control signals (does not concern Laser Mode 4 or Laser Mode 6). = 0: No phase shift. = 1: CO₂ Mode: The LASER1 signal is exchanged with the LASER2 signal. YAG modes: The LASER1 is shifted back half a signal period.</p> <p>Bit #2 Enabling or disabling of Signals for “Laser Active” Operation. = 0: The Signals for “Laser Active” Operation are enabled. = 1: The Signals for “Laser Active” Operation are disabled (the signals are set to their respective “Off” level).</p> <p>Bit #3 Level of LASERON signal. = 0: Is set to active-HIGH. = 1: Is set to active-LOW. If there is no change of the pin assignment with config_laser_signals, the LASERON signal corresponds to the signal at the LASERON pin, see Figure 17.</p> <p>Bit #4 Levels of LASER1 signal and LASER2 signal. = 0: Are set to active-HIGH. = 1: Are set to active-LOW. If there is no change of the pin assignment with config_laser_signals, the LASER1 signal (LASER2 signal) corresponds to the signal at the LASER1 pin (LASER2 pin), see Figure 17.</p> <p>Bit #5 Determines for laser_on_pulses_list whether external signal pulses (at the LASER Connector’s DIGITAL IN1 digital input port) are to be counted at rising or falling edges: = 0: At the falling edge. = 1: At the rising edge.</p> <p>Bit #6 = 0: Synchronization is switched off (default setting). See Chapter 7.4.10 “Synchronization of the RTC6 Clock Cycle and an External Clock Signal”, page 206. = 1: Synchronization is switched on.</p>

Ctrl Command	set_laser_control
Parameters (cont'd)	<p>Bit #7 = 0: The “constant pulse length” mode is switched off (default setting).</p> <p>= 1: The “constant pulse length” mode is switched on. See Chapter 7.4.8 “Pulse Picking Laser Mode”, page 195 and set_pulse_picking_length.</p> <p>Bit #8 Reserved.</p> <p>...</p> <p>Bit #15 Reserved.</p> <p>Bit #16 For the automatic suppression of laser control signals is used: PowerOK of the head A, x axis.</p> <p>Bit #17 Like Bit #16, but: TempOK of the head A, x axis.</p> <p>Bit #18 Like Bit #16, but: PosAck of the head A, x axis.</p> <p>Bit #19 Like Bit #16, but: PowerOK of the head A, y axis.</p> <p>Bit #20 Like Bit #16, but: TempOK of the head A, y axis.</p> <p>Bit #21 Like Bit #16, but: PosAck of the head A, y axis.</p> <p>Bit #22 Like Bit #16, but: PowerOK of the head B, x axis.</p> <p>Bit #23 Like Bit #16, but: TempOK of the head B, x axis.</p> <p>Bit #24 Like Bit #16, but: PosAck of the head B, x axis.</p> <p>Bit #25 Like Bit #16, but: PowerOK of the head B, y axis.</p> <p>Bit #26 Like Bit #16, but: TempOK of the head B, y axis.</p> <p>Bit #27 Like Bit #16, but: PosAck of the head B, y axis.</p> <p>Bit #28 = 1: In case of error, automatic monitoring (automatic suppression of laser control signals) automatically generates a /STOP signal (list stops, laser control signals get permanently switched off).</p> <p>Bit #29 = 1: In case of error according to Bit #28, the stop_execution is forwarded as /Master-STOP (see Figure 68) to all Master/Slave-connected RTC6 boards. See master_slave_config.</p> <p>Bit #30 Reserved.</p> <p>Bit #31 Reserved.</p>
Comments	<ul style="list-style-type: none"> In the default setting (after load_program_file), all bits are set to 0. After a hardware reset, however, the settings become effective following the first-time call of set_laser_control. Prior to this, all laser control signal outputs (LASERON, LASER1 and LASER2) are in the high-impedance mode. TTL states (LOW or HIGH) only become available when set_laser_control is called to define the desired TTL level, see also Chapter 7.4 “Laser Control”, page 183. Even after load_program_file, which deactivates the laser control signals, set_laser_control must be called for first-time activation.



Ctrl Command	set_laser_control
Comments (cont'd)	<ul style="list-style-type: none"> For the RTC5/RTC6 predecessors, the laser control signal levels are defined by solder jumpers. The RTC5/RTC6 lets users control them completely by software. 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 enable_laser or disable_laser. By get_startstop_info (Bit #14) the current state can be queried. Even if the laser control signals were enabled with set_laser_control or enable_laser, they are not outputted without further commands, see Chapter 7.4 "Laser Control", page 183. The phase shift of the laser control signals (Bit #1 = 1) can be set for better synchronization of an analog output, for example, in Softstart Mode (not yet implemented) or the Pixel Output Mode, see Chapter 8.7 "Pixel Output Mode", page 262.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 617, OUT 617, RBF 623: get_startstop_info (Bit #14) support.
References	set_laser_mode , config_laser_signals

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	<p>LaserOnDelay LaserOn Delay. As a signed 32-bit value. 1 bit equals $1/64 \mu\text{s}$. Allowed value range: $[-2^{31} \dots + (2^{21}-1)]$. Values over $(2^{21}-1)$ are clipped.</p> <p>LaserOffDelay LaserOff Delay. As an unsigned 32-bit value. 1 bit equals $1/64 \mu\text{s}$. Allowed value range: $[0 \dots + (2^{21}-1)]$. Values over $(2^{21}-1)$ are clipped.</p>
	<ul style="list-style-type: none"> 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 Section "Automatic Delay Adjustments", page 154. Observe the notes in Chapter 7.2.1 "Laser Delays", page 144. A negative <code>LaserOnDelay</code> value extends the total marking time. The XY2-100 Converter (Accessory) introduces a $10 \mu\text{s}$ runtime latency to scan system control. This runtime latency can be compensated by increasing the LaserOn Delay and LaserOff Delay by $10 \mu\text{s}$ each. The default setting after <code>load_program_file</code> corresponds to <code>set_laser_delays(640, 640)</code>.
RTC4→RTC6	Essentially similar functionality. In addition: increased value range. In RTC4 Compatibility Mode , the RTC6 multiplies the specified delay values by 64. The allowed value ranges decrease accordingly.
RTC5→RTC6	Unchanged functionality. In RTC5 Compatibility Mode , the RTC6 multiplies the values specified for <code>LaserOnDelay</code> and <code>LaserOffDelay</code> by 32. The allowed value ranges decrease accordingly.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_scanner_delays

Ctrl Command	set_laser_mode
Function	Sets the laser mode of the RTC6.
Call	<code>set_laser_mode(Mode)</code>
Parameters	<p>Mode = 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.</p> <p>As an unsigned 32-bit value.</p>
Comments	<ul style="list-style-type: none"> The available laser control signals depend on the set laser mode, see also Chapter 7.4 "Laser Control", page 183.
RTC4→RTC6	Additional laser modes, for example, YAG Mode 5, Laser Mode 6 and Pulse Picking. Otherwise, unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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	Sets the default output value for the ANALOG OUT1 and ANALOG OUT2 output ports, as well as for the 8-bit digital output port of the RTC6.
Call	<code>set_laser_off_default(AnalogOut1, AnalogOut2, DigitalOut)</code>
Parameters	<p>AnalogOut1 12-bit value for analog output port ANALOG OUT1, see also Section "12-Bit Analog Output Port 1 and 2", page 73. As an unsigned 32-bit value. Higher bits are ignored (exception: AnalogOut1/AnalogOut2 = "-1", see comments for set_port_default).</p> <p>AnalogOut2 12-bit value for analog output port ANALOG OUT2. See AnalogOut1.</p> <p>DigitalOut 8-bit value for the 8-bit digital output port, see also Section "8-Bit Digital Output Port", page 80. 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 set_port_default. See also all comments there.
RTC4→RTC6	<p>New command.</p> <p>In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for AnalogOut1 and AnalogOut2 by 4.</p>
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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>
Parameters	<p>Pins Output value (DIGITAL OUT1 and DIGITAL OUT2). As an unsigned 32-bit value.</p> <p>Bit # 0: DIGITAL OUT1.</p> <p>Bit # 1: DIGITAL OUT2.</p> <p>Bit # 2: Reserved.</p> <p>...</p> <p>Bit #31: Reserved.</p>
Comments	<ul style="list-style-type: none"> • See also Chapter 9.1.3 "2 Bit Digital Output Port", page 282.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_laser_pin_out_list , get_laser_pin_in

Undelayed Short List Command	set_laser_pin_out_list
Function	Like set_laser_pin_out , but a list command.
Call	<code>set_laser_pin_out_list(Pins)</code>
Parameters	Pins Like set_laser_pin_out .
Comments	<ul style="list-style-type: none"> • See set_laser_pin_out.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_laser_pin_out , write_port_list



Undelayed Short List Command	set_laser_power				
Function	Synchronizes laser power outputs at the output ports (ANALOG OUT1 , ANALOG OUT2 , 8-bit digital output port, 16-bit digital output port) and Laser Delays .				
Call	set_laser_power(Port, Power)				
Parameters	<table> <tr> <td>Port</td><td>Output port for which a laser power value Power is to be defined. As an unsigned 32-bit value. Allowed values: = 0: ANALOG OUT1 output port. = 1: ANALOG OUT2 output port. = 2: 8-bit digital output port (EXTENSION 2 Socket Connector). = 3: 16-bit digital output port (EXTENSION 1 Socket Connector). > 3: set_laser_power is not executed (get_last_error return code RTC6_PARAM_ERROR). </td></tr> <tr> <td>Power</td><td>Desired laser power value. As an unsigned 32-bit value. Allowed values: For Port = 0: 12-bit values [0...4095]. For Port = 1: 12-bit values [0...4095]. For Port = 2: 8-bit values [0...255]. For Port = 3: 16-bit values [0...65,535]. Out-of-range values are clipped to the boundary values. get_last_error return code RTC6_PARAM_ERROR. </td></tr> </table>	Port	Output port for which a laser power value Power is to be defined. As an unsigned 32-bit value. Allowed values: = 0: ANALOG OUT1 output port. = 1: ANALOG OUT2 output port. = 2: 8-bit digital output port (EXTENSION 2 Socket Connector). = 3: 16-bit digital output port (EXTENSION 1 Socket Connector). > 3: set_laser_power is not executed (get_last_error return code RTC6_PARAM_ERROR). 	Power	Desired laser power value. As an unsigned 32-bit value. Allowed values: For Port = 0: 12-bit values [0...4095]. For Port = 1: 12-bit values [0...4095]. For Port = 2: 8-bit values [0...255]. For Port = 3: 16-bit values [0...65,535]. Out-of-range values are clipped to the boundary values. get_last_error return code RTC6_PARAM_ERROR .
Port	Output port for which a laser power value Power is to be defined. As an unsigned 32-bit value. Allowed values: = 0: ANALOG OUT1 output port. = 1: ANALOG OUT2 output port. = 2: 8-bit digital output port (EXTENSION 2 Socket Connector). = 3: 16-bit digital output port (EXTENSION 1 Socket Connector). > 3: set_laser_power is not executed (get_last_error return code RTC6_PARAM_ERROR). 				
Power	Desired laser power value. As an unsigned 32-bit value. Allowed values: For Port = 0: 12-bit values [0...4095]. For Port = 1: 12-bit values [0...4095]. For Port = 2: 8-bit values [0...255]. For Port = 3: 16-bit values [0...65,535]. Out-of-range values are clipped to the boundary values. get_last_error return code RTC6_PARAM_ERROR . 				
Comments	<ul style="list-style-type: none"> In particular, the set_laser_power command is to be used with excelliSCAN scan heads, if laser power needs to be changed within a Polyline. The set_laser_power command is also necessary, for example, if short vectors need unequal laser power and Laser Delays extend beyond the next vector (in particular in DSP mode 3). 				
RTC4→RTC6	<p>New command.</p> <p>In RTC4 Compatibility Mode, the RTC6 multiplies the specified Power value for Port = 0 and Port = 1 by 4. The allowed value range decreases accordingly.</p>				
RTC5→RTC6	New command.				
Version info	Available as of DLL 602, OUT 602, RBF 602.				
References	get_last_error , set_dsp_mode , set_rtc4_mode , write_da_x				



Ctrl Command	set_laser_pulse_sync
Function	Switches Pulse Synchronization Mode on (off).
Call	set_laser_pulse_sync(Mode, Delay)
Parameters	Mode = 0: Switches Pulse Synchronization Mode off. > 0: Switches Pulse Synchronization Mode on. As an unsigned 32-bit value.
	Delay Delay. In 1/64 μ s. As an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> • See also Chapter 7.4.11 "Pulse Synchronization Mode", page 207. • See also Section ""Spot Distance Control"", page 201. • set_laser_pulse_sync is not executed (get_last_error return code RTC6_BUSY), if: <ul style="list-style-type: none"> – the BUSY list execution status is set – the INTERNAL-BUSY list execution status is set
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 625 , OUT 625 , RBF 628 .
References	–

Delayed Short List Command	set_laser_pulses
Function	Defines the output period and the pulse lengths of the laser control signals LASER1 and LASER2 for “laser active” operation.
Call	<code>set_laser_pulses(HalfPeriod, PulseLength)</code>
Parameters	<p><code>HalfPeriod</code> <i>Half of the output period. In bits. As an unsigned 32-bit value. 1 bit equals 1/64 μs. Allowed value range: [0...(2³²-1)].</i></p> <p><code>PulseLength</code> <i>Pulse length of the laser control signals LASER1 and LASER2. In bits. As an unsigned 32-bit value. 1 bit equals 1/64 μs. Allowed value range: [0...(2³²-1)].</i></p>
	<ul style="list-style-type: none"> • By the <code>HalfPeriod</code> parameter, <i>half</i> the period duration is specified, see Figure 52 and Figure 53. • If <code>HalfPeriod = 0</code> and/or <code>PulseLength = 0</code>, no laser control signals are outputted. • With <code>PulseLength \geq (2 \times HalfPeriod)</code>, the laser remains on all the time. • The signal level is defined by set_laser_control. • set_laser_pulses is also available as the control command set_laser_pulses_ctrl. • set_laser_pulses is largely identical to the RTC4 command set_laser_timing, but has less parameters.
RTC4→RTC6	New command. In RTC4 Compatibility Mode , the RTC6 multiplies the specified values for <code>HalfPeriod</code> and <code>PulseLength</code> by 8. The allowed value ranges decrease accordingly.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_laser_pulses_ctrl , set_laser_timing



Ctrl Command	set_laser_pulses_ctrl
Function	Like set_laser_pulses , but a control command.
Call	<code>set_laser_pulses_ctrl(HalfPeriod, PulseLength)</code>
Parameters	HalfPeriod Like set_laser_pulses .
	PulseLength Like set_laser_pulses .
Comments	<ul style="list-style-type: none">• See set_laser_pulses.
RTC4→RTC6	New command. In RTC4 Compatibility Mode : like set_laser_pulses .
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 control signals LASER1 and LASER2 for “laser active” operation.	
Call	set_laser_timing(<i>HalfPeriod</i> , <i>PulseLength1</i> , <i>(PulseLength2)</i> , <i>TimeBase</i>)	
Parameters	<i>HalfPeriod</i> Half of the output period. In bits. As an unsigned 32-bit value. <ul style="list-style-type: none"> • In RTC6 Standard Mode: 1 bit equals 1/64 μs. Allowed value range: [0...(2³²-1)]. • In RTC4 Compatibility Mode: 1 bit equals 1/8 μs or 1 μs, depending on the selected clock frequency. With respect to the specified <i>TimeBase</i>, the value is converted to an integer-multiple of 1/64 μs. The allowed range is correspondingly smaller. 	
	<i>PulseLength1</i> Pulse lengths of the laser control signals LASER1 and LASER2. In bits. As an unsigned 32-bit value. <ul style="list-style-type: none"> • In RTC6 Standard Mode: 1 bit equals 1/64 μs. Allowed value range: [0...(2³²-1)]. • In RTC4 Compatibility Mode: 1 bit equals 1/8 μs or 1 μs, depending on the selected clock frequency. With respect to the specified <i>TimeBase</i>, the value is converted to an integer-multiple of 1/64 μs. The allowed range is correspondingly smaller. 	
	<i>(PulseLength2)</i> Value is not used. (As an unsigned 32-bit value.) With the RTC6, the pulse lengths of laser control signals LASER1 and LASER2 are always identical and are defined by <i>PulseLength1</i> .	
	<i>TimeBase</i> As an unsigned 32-bit value. <ul style="list-style-type: none"> • In RTC6 Standard Mode, the value is ignored and the clock frequency is fixed at 64 MHz. 1 bit equals 1/64 μs. • In RTC4 Compatibility Mode, the <i>TimeBase</i> value is handled as follows: = 0: sets the clock frequency to 1 MHz. 1 bit equals 1 μs. ≠ 0: sets the clock frequency to 8 MHz. 1 bit equals 1/8 μs. 	

Delayed Short List Command	set_laser_timing
Comments	<ul style="list-style-type: none"> In RTC6 Standard Mode, set_laser_timing is synonymous with set_laser_pulses. See comments there (on <code>HalfPeriod</code>, <code>PulseLength</code>). The clock frequency settings apply <i>only</i> for the parameters of set_laser_timing. In RTC4 Compatibility Mode, SCANLAB generally recommends setting the clock frequency to 8 MHz. A clock frequency of 1 MHz should only be set, if absolutely necessary. Observe also the notes in Chapter 7.4 "Laser Control", page 183. In RTC4 Compatibility Mode, in Laser Mode 4 and Laser Mode 6, the time base for signals LASER1 and LASER2 is independently of <code>TimeBase</code> always 1/8 μs.
RTC4→RTC6	<ul style="list-style-type: none"> With the RTC6, the pulse lengths of laser control signals LASER1 and LASER2 are always identical. Clock frequencies: <ul style="list-style-type: none"> In RTC6 Standard Mode: 64 MHz, fixed In RTC4 Compatibility Mode: 1 MHz or 8 MHz
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_laser_pulses_ctrl , set_laser_pulses , set_laser_mode , set_firstpulse_killer , set_firstpulse_killer_list , set_standby , set_standby_list

Normal List Command	set_laser_timing_table				
Function	<p>Category: List Commands for a Restricted User Group only.</p> <p>Sets a value in the lookup table for the laser frequency.</p>				
Call	<code>set_laser_timing_table(Index, F)</code>				
Parameters	<table> <tr> <td>Index</td> <td>Index in the lookup table for the laser frequency. Allowed value range: 0...10. As an unsigned 32-bit value.</td> </tr> <tr> <td>F</td> <td>Laser frequency. In Hz. Allowed value range: 65...100,000. As an unsigned 32-bit value.</td> </tr> </table>	Index	Index in the lookup table for the laser frequency. Allowed value range: 0...10. As an unsigned 32-bit value.	F	Laser frequency. In Hz. Allowed value range: 65...100,000. As an unsigned 32-bit value.
Index	Index in the lookup table for the laser frequency. Allowed value range: 0...10. As an unsigned 32-bit value.				
F	Laser frequency. In Hz. Allowed value range: 65...100,000. As an unsigned 32-bit value.				
Comments	<ul style="list-style-type: none"> All Comments of set_duty_cycle_table apply. 				
RTC4→RTC6	New command.				
RTC5→RTC6	New command.				
Version info	Available as of DLL 626, OUT 627.				
References	set_duty_cycle_table , regulation3				



Undelayed Short List Command	set_list_jump
Function	Produces an unconditional jump to the specified address within the list memory upon execution. The next command there is executed immediately without delay.
Call	<code>set_list_jump(Pos)</code>
Parameters	Pos Absolute jump address [0...(2 ²³ -1)]. As an unsigned 32-bit value.
Comments	• set_list_jump is synonymous with list_jump_pos . See the comments there.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	list_jump_pos

Delayed Short List Command	set_mark_speed
Function	Sets the mark speed for the Vector commands and "Arc" commands .
Call	<code>set_mark_speed(Speed)</code>
Parameters	Speed Marking speed. In Bits/ms. As a 64-bit IEEE floating point value. Allowed value range: [1.6...800000.0].
Comments	<ul style="list-style-type: none"> By default a mark speed of 1,000 <i>bits/ms</i> is preset. The specified mark speed is used for all [*]mark[*] Commands and "Arc" commands until a new value is specified. The actual mark speed v_{mark} in the image plane in m/s is derived from the specified Speed value [<i>bits/ms</i>] and the calibration factor K [Bits/mm] as follows: $v_{mark} = \text{Speed} / K$ <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"> set_mark_speed is also available as control command set_mark_speed_ctrl.
RTC4→RTC6	Unchanged functionality. In RTC4 Compatibility Mode , the RTC6 multiplies the specified Speed value by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	mark_abs, mark_rel, set_jump_speed, set_mark_speed_ctrl

Ctrl Command	set_mark_speed_ctrl
Function	Like set_mark_speed , but a control command.
Call	<code>set_mark_speed_ctrl(Speed)</code>
Parameters	Speed Like set_mark_speed .
Comments	<ul style="list-style-type: none"> set_mark_speed_ctrl is not executed (get_last_error return code RTC6_BUSY), if: <ul style="list-style-type: none"> the BUSY list execution status is set set_mark_speed_ctrl is even executed, if: <ul style="list-style-type: none"> a list has been paused by set_wait (PAUSED list execution status set) the INTERNAL-BUSY list execution status is set
RTC4→RTC6	New command. In RTC4 Compatibility Mode : like set_mark_speed .
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_mark_speed, set_jump_speed, set_jump_speed_ctrl

Ctrl Command	set_matrix
Function	Defines the coefficients of the <ul style="list-style-type: none"> General transformation matrix M_T for coordinate transformations, see Chapter 8.2 "Coordinate Transformations", page 223 Global transformation matrix in virtual Image field, see Section "Coordinate Transformations in the Virtual Image Field", page 168
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>first</i> scan head connector. = 2: The definition only affects the <i>second</i> scan head connector. = 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 Matrix coefficient of the general transformation matrix M_T. As a 64-bit IEEE floating point value. Allowed value range: [-50...+50] in the real Image field and [-2.0...+2.0] in the virtual Image field. With invalid value, set_matrix is ignored.</p> <p>M12 Like M11 (analogously).</p> <p>M21 Like M11 (analogously).</p> <p>M22 Like M11 (analogously).</p> <p>at_once Defines when the defined transformation becomes effective. As an unsigned 32-bit value.</p> <p>For HeadNo = 0...3, the following applies:</p> <ul style="list-style-type: none"> = 0: The new total transformation (total matrix and offset) is only calculated when the next list command is executed and applied to the position which is current at that time. = 1: The new total transformation is calculated immediately (or prior the next list command, if currently the BUSY list execution status or INTERNAL-BUSY list execution status is set) and applied to the current position. Signals for "Laser Active" Operation are switched off in advance. = 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 is executed. = 3: Like at_once = 1, but the laser control signals remain unchanged. > 3: Like at_once = 2. <p>For HeadNo = 4, the following applies:</p> <ul style="list-style-type: none"> = 0: Only saved. Is applied at the next Processing-on-the-fly session start to the current position. = 1: Is applied immediately to the current position if currently no list is BUSY list execution status or no board is INTERNAL-BUSY list execution status. Otherwise, like 0. = 2, 3: Like 0.



Ctrl Command	set_matrix
Comments	<ul style="list-style-type: none"> Up to DLL 613, OUT 613 the following applies for HeadNo = 4: <ul style="list-style-type: none"> The global coordinate transformations are only available during a Processing-on-the-fly session that has been started by set_fly_2d. <code>at_once</code> is ignored As of DLL 614, OUT 614 the following applies for HeadNo = 4: <ul style="list-style-type: none"> The global coordinate transformations are generally available, even outside a Processing-on-the-fly session. Coordinate transformations that have been merely saved, are applied at the next Processing-on-the-fly session start.
RTC4→RTC6	Unchanged first functionality (transformation matrix definition), however: <ul style="list-style-type: none"> The parameters HeadNo and <code>at_once</code> are new. Reduced value range for coefficients. See also Section "Notes for RTC4 Users", page 226.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 614, OUT 614: HeadNo = 4 generally available.
References	set_matrix_list , set_angle , set_offset , set_scale

Variable List Command	set_matrix_list
Function	Sets one of the 4 coefficients of the general transformation matrix M_T during execution of a list, see Chapter 8.2 "Coordinate Transformations", page 223 .
Call	<code>set_matrix_list(HeadNo, Ind1, Ind2, Mij, at_once)</code>
Parameters	<p>HeadNo Like set_matrix.</p> <p>Ind1 Row index and column index of the matrix coefficient to be changed. As an unsigned 32-bit value.</p> <p>Ind2 Allowed values: [Index uneven: 1, Index even: 2].</p> <p>Mij Matrix coefficient. As a 64-bit IEEE floating point value. Allowed value range: [-50...+50]. If the parameter is set to an invalid value, set_matrix_list is replaced by a list_nop.</p> <p>at_once Determines when the defined transformation becomes effective. As an unsigned 32-bit value. = 0: The transformation settings are only accumulated and intermediately stored, but the transformation is not processed as long as this is not activated by another coordinate transformation (for example, 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 accumulated until then) and processed prior to the next list command. Signals for "Laser Active" Operation are switched off in advance. = 2: The transformation settings are only accumulated and intermediately stored (as with <code>at_once = 0</code>). However, The transformation is immediately calculated (including all transformation settings that were accumulated and intermediately stored until then) and applied to the current position when the next jump_abs or jump_rel (only if no list is currently being executed: also goto_xy or goto_xyz) 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"> • set_matrix_list only allows changing one of the 4 coefficients at a time. To change several coefficients during execution of a list, set_matrix_list 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 223.
RTC4→RTC6	See set_matrix .
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_matrix

Ctrl Command	set_max_counts
Function	Defines the maximum number of External Starts .
Call	<code>set_max_counts(Counts)</code>
Parameters	Counts Maximum number of External Starts . As an unsigned 32-bit value. Allowed value range: [0...(2 ³² -1)].
Comments	<ul style="list-style-type: none"> When the specified number of External Starts has been reached, the external start input is disabled (see set_control_mode, Bit #0 = 0). If Counts = 0, the number of External Starts is unlimited. When the RTC6 is initialized (by load_program_file), Counts is set to 0. The current number of External Starts can be read from the corresponding internal counter by get_counts. The counter can be reset by set_control_mode.
RTC4→RTC6	Unchanged functionality. In addition: increased value range.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	get_counts, set_control_mode

Ctrl Command	set_mcbsp_freq
Function	Sets the transmission frequency of the McBSP interface .
Call	<code>mcbsp_freq = set_mcbsp_freq(Freq)</code>
Parameters	Freq Desired transmission frequency of the McBSP interface in Hz. As an unsigned 32-bit value. Allowed value range: [4000000...16000000] (4...16 MHz). Out-of-range values are clipped to the boundary values. Out-of-range values cause the get_last_error return code RTC6_PARAM_ERROR to be generated.
Result	The actually set frequency in Hz. As an unsigned 32-bit value. With overflowing values 0 is returned.
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, set_mcbsp_freq 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 interface by mcbsp_init. The signals and operating conditions of the McBSP interface are presented in the Chapter 4.6.6 "McBSP/ANALOG Socket Connector", page 83. Note: The receiving frequency is exclusively determined by the incoming clock pulses and has a maximum limit of 16 MHz.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	mcbsp_init, set_mcbsp_out, set_mcbsp_out_ptr

Ctrl Command	<u>set_mcbsp_global_matrix</u>
Function	Activates matrix correction for “Global Online Positioning” by the McBSP interface .
Call	<code>set_mcbsp_global_matrix()</code>
Comments	<ul style="list-style-type: none"> See also Chapter 8.3.2 “Global Online Positioning”, page 230. A matrix correction cannot be used in conjunction with offset and/or rotation corrections. Any such already-activated options gets deactivated by set_mcbsp_global_matrix. Subsequent activation of other options (by set_mcbsp_global_x, set_mcbsp_global_y or set_mcbsp_global_rot) deactivates the matrix option. The following restrictions apply to the matrix coefficients transferred over the McBSP interface (as with set_matrix (<code>HeadNo = 4, ...</code>)): <ul style="list-style-type: none"> The allowed value range for matrix coefficients is $[-2.0 \dots +2.0]$. Transferred coefficients exceeding this range are ignored. Users must individually supply as input value M_{in} to the McBSP interface each matrix coefficient M_{ij} of the transformation matrix M_T as a normalized integer with associated indices i and j as follows: $M_{in} = (\text{integer}(M_{ij} * 2^{28}) << 2) + (i << 1) + j$ with $M_T = \{ M_{00}, M_{01}, M_{10}, M_{11} \} = \{ m_{11}, m_{12}, m_{21}, m_{22} \}$. Conversely, the RTC6 determines a coefficient from the input value as follows: $M_T[M_{in} \& 0x3] = (M_{in} >> 2) / 2^{28}$. The coefficients are transferred to internal memory location 1 and can be checked there by querying with read_mcbsp(1). The McBSP interface cannot be simultaneously used for an Online Positioning and Processing-on-the-fly applications. <p>See also Section “Notes”, page 229.</p>
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 617, OUT 617, RBF 623.
References	<u>set_mcbsp_global_matrix_list</u> , <u>set_mcbsp_matrix</u>

Undelayed Short List Command	<u>set_mcbsp_global_matrix_list</u>
Function	Like set_mcbsp_global_matrix , but a list command.
Call	<code>set_mcbsp_global_matrix_list()</code>
Comments	<ul style="list-style-type: none"> See set_mcbsp_global_matrix.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 617, OUT 617, RBF 623.
References	<u>set_mcbsp_global_matrix</u>

Ctrl Command	set_mcbsp_global_rot
Function	Activates or deactivates rotation correction for “Global Online Positioning” by the McBSP interface .
Call	set_mcbsp_global_rot(Resolution)
Parameters	<p>Resolution As a 64-bit IEEE floating point value.</p> <p>$2^{-26} < \text{Resolution} < 2^{26}$: scaling factor (correction is activated), otherwise: correction is deactivated.</p> <p>Resolution = McBSP bits per full circle.</p>
Comments	<ul style="list-style-type: none"> See also Chapter 8.3.2 “Global Online Positioning”, page 230. With an McBSP rotation correction input value of Rot_{in}, set_mcbsp_global_rot functions like <code>set_angle(4, Angle × 360°, ...)</code>, whereby Angle (in full circles) = $\text{Rot}_{\text{in}} / \text{Resolution}$. Only Angle values in the range [0.0...+20.0 full circles] are allowed. The McBSP interface cannot be simultaneously used for an Online Positioning and Processing-on-the-fly applications. <p>See also Section “Notes”, page 229.</p>
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 617, OUT 617, RBF 623.
References	set_mcbsp_global_rot_list , set_mcbsp_rot

Undelayed Short List Command	set_mcbsp_global_rot_list
Function	Like set_mcbsp_global_rot , but a list command.
Call	set_mcbsp_global_rot_list(Resolution)
Parameters	Resolution Like set_mcbsp_global_rot .
Comments	<ul style="list-style-type: none"> See set_mcbsp_global_rot.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 617, OUT 617, RBF 623.
References	set_mcbsp_global_rot

Ctrl Command	set_mcbsp_global_x
Function	Activates or deactivates x offset correction for “ Global Online Positioning ” by the McBSP interface .
Call	<code>set_mcbsp_global_x(Scale)</code>
Parameters	Scale As a 64-bit IEEE floating point value. $2^{-26} < \text{Scale} < 2^{26}$: scaling factor (correction is activated), otherwise: correction is deactivated.
Comments	<ul style="list-style-type: none"> • See also Chapter 8.3.2 “Global Online Positioning”, page 230. • With an McBSP input value of X_{in} for the x offset correction, set_mcbsp_global_x functions like <code>set_offset_xyz(4, XOffset,...)</code>, whereby $XOffset$ (in bits) = $\text{Scale} \times X_{in}$. $XOffset$ values outside of $[-524,288\dots+524,287]$ are clipped to the boundary value as long as $\text{Scale} \times X_{in}$ does not exceed the value range $\pm 2^{31}$. • The McBSP interface cannot be simultaneously used for an Online Positioning and Processing-on-the-fly applications. See also Section “Notes”, page 229.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 617, OUT 617, RBF 623.
References	set_mcbsp_global_x_list , set_mcbsp_x

Undelayed Short List Command	set_mcbsp_global_x_list
Function	Like set_mcbsp_global_x , but a list command.
Call	<code>set_mcbsp_global_x_list(Scale)</code>
Parameters	Scale Like set_mcbsp_global_x .
Comments	<ul style="list-style-type: none"> • See set_mcbsp_global_x.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 617, OUT 617, RBF 623.
References	set_mcbsp_global_x

Ctrl Command	set_mcbsp_global_y
Function	Activates or deactivates y offset correction for “ Global Online Positioning ” by the McBSP interface.
Call	<code>set_mcbsp_global_y(Scale)</code>
Parameters	Scale As a 64-bit IEEE floating point value. $2^{-26} < \text{Scale} < 2^{26}$: scaling factor (correction is activated), otherwise: correction is deactivated.
Comments	<ul style="list-style-type: none"> See also Chapter 8.3.2 “Global Online Positioning”, page 230. With an McBSP input value of Y_{in} for the y offset correction, <code>set_mcbsp_global_y</code> functions like <code>set_offset_xyz(4, ..., YOffset,...)</code>, whereby Y_{Offset} (in bits) = $\text{Scale} \times Y_{in}$. Y_{Offset} values outside of $[-524,288...+524,287]$ are clipped to the boundary value as long as $\text{Scale} \times Y_{in}$ does not exceed the value range $\pm 2^{31}$. The McBSP interface cannot be simultaneously used for an Online Positioning and Processing-on-the-fly applications. See also Section “Notes”, page 229.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 617, OUT 617, RBF 623.
References	set_mcbsp_global_y_list , set_mcbsp_y

Undelayed Short List Command	set_mcbsp_global_y_list
Function	Like set_mcbsp_global_y , but a list command.
Call	<code>set_mcbsp_global_y_list(Scale)</code>
Parameters	Scale Like set_mcbsp_global_y .
Comments	<ul style="list-style-type: none"> See set_mcbsp_global_y.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 617, OUT 617, RBF 623.
References	set_mcbsp_global_y

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 RTC6 via the McBSP interface). The McBSP interface can also be used for inputting other desired signals.
Restriction	If the Option Processing-on-the-fly is not enabled, then set_mcbsp_in 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 As an unsigned 32-bit value. Allowed values: = 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 direction and y direction. = 4: Compensation of rotary movement. = 5: Processing-on-the-fly correction is switched off.</p> <ul style="list-style-type: none"> • Mode = 0...5: All McBSP input values are alternatingly copied to internal memory locations 1 and 2. • Mode = 1...5 (but not for Mode = 0): McBSP 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 $(RTC6)bits/(McBSP)bit$. Allowed value 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 value 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 8.6 "Processing-on-the-fly", page 241. • The various Processing-on-the-fly corrections cannot be arbitrarily combined, see Section "Overview", page 241.

Ctrl Command	set_mcbsp_in
Comments (cont'd)	<ul style="list-style-type: none"> For deactivating Processing-on-the-fly correction, see Chapter 8.6.5 "Deactivating Processing-on-the-fly Correction", page 250. 15 bits per axis (with sign) are effectively available for 2D correction (<code>Mode</code> = 3), whereby the x value is in the lower 16 bits of the "Bit #31 = 0"-coded McBSP input value and the y value in the upper 16 bits. The McBSP interface cannot be simultaneously used for both Processing-on-the-fly applications and Online Positioning. See also Section "Notes", page 229. The McBSP interface ignores the first FrameSync signal after a load_program_file or mcbsp_init. That is, data provided is not transmitted, see page 85. If <code>Mode</code> > 5, <code>set_mcbsp_in</code> is not executed (<code>get_last_error</code> return code <code>RTC6_PARAM_ERROR</code>). If an unallowed <code>Scale</code> parameter value is supplied (for example, <code>Scale</code> = 0), <code>set_mcbsp_in</code> behaves as with <code>Mode</code> = 0. Processing-on-the-fly corrections are compatible with the settings from Chapter 8.6.12 ""Fly Extension" Commands", page 258 as of RTC6 Software Package V1.6.1. They can also be subsequently overwritten, for example, with encoder-based Processing-on-the-fly corrections. The McBSP values are then not used for Processing-on-the-fly corrections. However, the McBSP memory transfer described above cannot be changed. If McBSP-based Processing-on-the-fly corrections other than those set according to the <code>Mode</code> are set, the user is responsible for transferring the McBSP data in the appropriate format. An encoder-based Processing-on-the-fly correction activated for the z axis is retained.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Last change DLL 617, OUT 617: compatibility with "Fly Extension" Commands .
References	set_mcbsp_in_list



Normal List Command	<code>set_mcbsp_in_list</code>				
Function	Like <code>set_mcbsp_in</code> , but a list command.				
Restriction	If the Option Processing-on-the-fly is not enabled, then <code>set_mcbsp_in_list</code> 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>Like <code>set_mcbsp_in</code>.</td></tr> <tr> <td>Scale</td><td>Like <code>set_mcbsp_in</code>.</td></tr> </table>	Mode	Like <code>set_mcbsp_in</code> .	Scale	Like <code>set_mcbsp_in</code> .
Mode	Like <code>set_mcbsp_in</code> .				
Scale	Like <code>set_mcbsp_in</code> .				
Comments	<ul style="list-style-type: none"> If <code>Mode > 5</code>, then <code>set_mcbsp_in_list</code> is replaced by a <code>list_nop</code> (<code>get_last_error</code> return code <code>RTC6_PARAM_ERROR</code>). See <code>set_mcbsp_in</code>. 				
RTC4→RTC6	New command.				
RTC5→RTC6	Unchanged functionality.				
Version info	Last change DLL 617, OUT 617: compatibility with "Fly Extension" Commands .				
References	<code>set_mcbsp_in</code>				

Ctrl Command	set_mcbsp_matrix
Function	Activates matrix correction for "Local Online Positioning" by the McBSP interface.
Call	<code>set_mcbsp_matrix()</code>
Comments	<ul style="list-style-type: none"> For "Local Online Positioning", see Chapter 8.3.1 ""Local Online Positioning"", page 227. 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 correction. The following restrictions apply to the matrix coefficients transferred over the McBSP interface (as with set_matrix): <p>The allowed value range for matrix coefficients is $[-50 \dots +50]$. Transferred coefficients exceeding this range are ignored.</p> You must individually supply as input value M_{in} to the McBSP 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 RTC6 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 interface cannot be simultaneously used for an Online Positioning and Processing-on-the-fly applications. <p>See also Section "Notes", page 229.</p>
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_mcbsp_matrix_list



Undelayed Short List Command	set_mcbsp_matrix_list
Function	Like 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→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_mcbsp_matrix

Undelayed Short List Command	set_mcbsp_out
Function	Defines two signal types for output at the McBSP interface , see also Chapter 4.6.6 "McBSP/ANALOG Socket Connector", page 83 .
Call	<code>set_mcbsp_out(Signal1, Signal2)</code>
Parameters	Signal1 To-be-outputted signal type. As an unsigned 32-bit value.
	Signal2 To-be-outputted signal type. As an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> The selectable signal types are identical to those of set_trigger (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 RTC6_PARAM_ERROR). Both selected data signals are continuously transmitted (once per $10 \mu\text{s}$ 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 Bit #4...Bit #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...57), the least significant 16 bits are outputted and the upper bits are ignored. McBSP_Output = ((Out2 & 0x0000FFFF) << 16) (Out1 & 0x0000FFFF); 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 McBSP interface are presented in Chapter 4.6.6 "McBSP/ANALOG Socket Connector", page 83.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	read_mcbsp , mcbsp_init , set_mcbsp_freq , set_mcbsp_out_ptr , set_trigger

Ctrl Command	set_mcbsp_out_oie_ctrl
Function	Switches on the OIE Output Mode at the McBSP interface .
Call	set_mcbsp_out_oie_ctrl(Signal1, Signal2)
Parameters	Signal1 To-be-outputted signal type. As an unsigned 32-bit value.
	Signal2 To-be-outputted signal type. As an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> OIE Output Mode At the McBSP interface, a 32-bit data word of the following structure is outputted permanently (= once every 10 µs clock cycle): <ul style="list-style-type: none"> – Bit #00...Bit #11 Free variable 0, clipped to 12 bit. – Bit #12...Bit #13 0. – Bit #14...Bit #21 Signal1, scaled to [-128...+127]. – Bit #22...Bit #29 Signal2, scaled to [-128...+127]. – Bit #30 BUSY list execution status, siehe get_status. – Bit #31 LASERON signal. The selected signal types for Signal1 and Signal2 are identical to set_trigger. A valid position signal (with value range [-524,288...+524,287]) should be selected for both signal types. set_mcbsp_out_oie_ctrl is not executed with an unallowed parameter value (get_last_error return code RTC6_PARAM_ERROR).
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 633, OUT 634.
References	set_mcbsp_out_oie_list

Undelayed Short List Command	set_mcbsp_out_oie_list
Function	Like set_mcbsp_out_oie_ctrl , but a list command.
Call	set_mcbsp_out_oie_list(Signal1, Signal2)
Parameters	Signal1 Like set_mcbsp_out_oie_ctrl .
	Signal2 Like set_mcbsp_out_oie_ctrl .
Comments	<ul style="list-style-type: none"> See set_mcbsp_out_oie_ctrl. With an unallowed parameter value, set_mcbsp_out_oie_list is replaced by a list_nop (get_last_error return code RTC6_PARAM_ERROR).
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 633, OUT 634.
References	set_mcbsp_out_oie_ctrl

Ctrl Command	<code>set_mcbsp_out_ptr</code>
Function	Defines a list of up to 8 signal types for output at the McBSP interface , see Chapter 4.6.6 "McBSP/ANALOG Socket Connector", page 83 .
Call	<code>set_mcbsp_out_ptr(Number, SignalPtr)</code>
Parameters	<p>Number As an unsigned 32-bit value. Allowed value range: [0...8]. = 1...8: Number of signal types to be outputted. = 0 Output at the McBSP interface occurs in accordance with the pre-defined settings or as specified by a prior set_mcbsp_out.</p> <p>Bit 31 = 0: Signals are outputted as 24-bit values. Bit 31 = 1: Signals are outputted as 23-bit values. The most significant bit always contains the LASERON status.</p> <p>SignalPtr Pointer (in C and C++ data type <code>ULONG_PTR</code>, an unsigned 32-bit value or unsigned 64-bit value) to an array of <code>Number</code> unsigned 32-bit values, where the to-be-outputted <code>Number</code> signal type numbers are specified.</p>
Comments	<ul style="list-style-type: none"> The memory area for the <code>SignalPtr</code> array must be provided by the user program. If <code>Number > 8</code> and/or <code>SignalPtr = NULL</code>, <code>set_mcbsp_out_ptr</code> is not executed (<code>get_last_error</code> return code <code>RTC6_PARAM_ERROR</code>). The selectable signal types are identical to those of set_trigger (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. The list is repeated until the output is switched off or replaced by another list. 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 24, 31 and 37...57): $\text{32-bit output value} = (\text{signal value} \ll 8) \mid (\text{signal type number} \& 0xFF).$ The signals and operating conditions of the McBSP interface are presented in Chapter 4.6.6 "McBSP/ANALOG Socket Connector", page 83. Number Bit 31 is needed for the OIE PID control, for example. The following applies to Number-Bit 31 = 1: At the McBSP interface, the LASERON status is outputted 10 µs later than as recorded by set_trigger(<code>Signal1 = 0</code>).
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 632 , OUT 633 . Number Bit 31.
References	set_mcbsp_out_ptr_list , set_mcbsp_out , set_trigger



Multiple List Command	set_mcbsp_out_ptr_list				
Function	Like set_mcbsp_out_ptr , but a list command.				
Call	<code>set_mcbsp_out_ptr_list(Number, SignalPtr)</code>				
Parameters	<table> <tr> <td>Number</td> <td>Like set_mcbsp_out_ptr.</td> </tr> <tr> <td>SignalPtr</td> <td>Like set_mcbsp_out_ptr.</td> </tr> </table>	Number	Like set_mcbsp_out_ptr .	SignalPtr	Like set_mcbsp_out_ptr .
Number	Like set_mcbsp_out_ptr .				
SignalPtr	Like set_mcbsp_out_ptr .				
Comments	<ul style="list-style-type: none"> • set_mcbsp_out_ptr_list requires two list memory positions for <code>Number</code> ≥ 1. Sequence of execution: <ol style="list-style-type: none"> 1. Pending delayed short list commands 2. First command part as an undelayed short list command 3. Second command part as a normal list command • See set_mcbsp_out_ptr. 				
RTC4→RTC6	New command.				
RTC5→RTC6	Unchanged functionality.				
Version info	Available as of DLL 632 , OUT 633 .				
References	set_mcbsp_out_ptr				

Ctrl Command	set_mcbsp_rot
Function	Activates or deactivates rotation correction for "Local Online Positioning" by the McBSP interface .
Call	<code>set_mcbsp_rot(Resolution)</code>
Parameters	<p>Resolution As a 64-bit IEEE floating point value.</p> $2^{-26} < \text{Resolution} < 2^{26}$: scaling factor (correction is activated), otherwise: correction is deactivated. <p>Resolution = McBSP bits per full circle.</p>
Comments	<ul style="list-style-type: none"> For "Local Online Positioning", see Chapter 8.3.1 ""Local Online Positioning"", page 227. For an McBSP rotation correction input value of Rot_{in}, apply_mcbsp functions like <code>set_angle(..., Angle × 360°, ...)</code>, whereby $\text{Angle in full circles} = \text{Rot}_{\text{in}} / \text{Resolution}$ <p>Only <code>Angle</code> values in the range $[0.0 \dots +20.0$ full circles] are allowed.</p> <ul style="list-style-type: none"> The McBSP interface cannot be simultaneously used for an Online Positioning and Processing-on-the-fly applications. <p>See also Section "Notes", page 229.</p>
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_angle , set_mcbsp_rot_list , set_mcbsp_x , set_mcbsp_y , apply_mcbsp

Undelayed Short List Command	set_mcbsp_rot_list
Function	Like set_mcbsp_rot , but a list command.
Call	<code>set_mcbsp_rot_list(Resolution)</code>
Parameters	Resolution Like set_mcbsp_rot .
Comments	<ul style="list-style-type: none"> See set_mcbsp_rot.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_mcbsp_rot



Ctrl Command	set_mcbsp_x
Function	Activates or deactivates X offset correction for "Local Online Positioning" by the McBSP interface .
Call	<code>set_mcbsp_x(Scale)</code>
Parameters	Scale As a 64-bit IEEE floating point value. $2^{-26} < \text{Scale} < 2^{26}$: scaling factor (correction is activated). Otherwise: correction is deactivated.
Comments	<ul style="list-style-type: none"> For "Local Online Positioning", see Chapter 8.3.1 ""Local Online Positioning"", page 227. With an McBSP input value of X_{in} for the X offset correction, apply_mcbsp functions like <code>set_offset(..., XOffset, ...)</code>, whereby $XOffset \text{ (in bits)} = \text{Scale} \times X_{in}$ $XOffset \text{ values outside of } [-524,288 \dots +524,287] \text{ are clipped to the boundaries as long as } \text{Scale} \times X_{in} \text{ does not exceed the value range } \pm 2^{31}.$ The McBSP interface cannot be simultaneously used for an Online Positioning and Processing-on-the-fly applications. See also Section "Notes", page 229.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_mcbsp_x_list , set_mcbsp_y , set_mcbsp_rot , apply_mcbsp

Undelayed Short List Command	set_mcbsp_x_list
Function	Like set_mcbsp_x , but a list command.
Call	<code>set_mcbsp_x_list(Scale)</code>
Parameters	Scale Like set_mcbsp_x .
Comments	<ul style="list-style-type: none"> See set_mcbsp_x.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_mcbsp_x



Ctrl Command	set_mcbsp_y
Function	Activates or deactivates y offset correction for “ Local Online Positioning ” by the McBSP interface .
Call	set_mcbsp_y(Scale)
Parameters	Scale As a 64-bit IEEE floating point value. $2^{-26} < \text{Scale} < 2^{26}$: scaling factor (correction is activated). Otherwise: correction is deactivated.
Comments	<ul style="list-style-type: none"> For “Local Online Positioning”, see Chapter 8.3.1 ““Local Online Positioning””, page 227. With an McBSP input value of Y_{in} for the y offset correction, apply_mcbsp functions like <code>set_offset(..., YOffset,...)</code>, whereby $YOffset \text{ (in bits)} = \text{Scale} \times Y_{in}$ $YOffset \text{ values outside of } [-524,288 \dots +524,287] \text{ are clipped to the boundaries as long as } \text{Scale} \times Y_{in} \text{ does not exceed the value range } \pm 2^{31}.$ The McBSP interface cannot be simultaneously used for an Online Positioning and Processing-on-the-fly applications. See also Section “Notes”, page 229.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_mcbsp_y_list , set_mcbsp_x , set_mcbsp_rot , apply_mcbsp

Undelayed Short List Command	set_mcbsp_y_list
Function	Like set_mcbsp_y , but a list command.
Call	set_mcbsp_y_list(Scale)
Parameters	Scale Like set_mcbsp_y .
Comments	<ul style="list-style-type: none"> See set_mcbsp_y.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_mcbsp_y

Ctrl Command	set_multi_mcbsp_in
Function	Activates a Processing-on-the-fly application and McBSP interface multi transmission with up to 8 different data types.
Restriction	If the Option Processing-on-the-fly is not enabled, then set_multi_mcbsp_in switches off the Processing-on-the-fly process (even if it has been never switched on).
Call	<code>set_multi_mcbsp_in(Ctrl, P, Mode)</code>
Parameters	<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 RTC6_PARAM_ERROR). P Initialization value for laser power. As an unsigned 32-bit value. Allowed value range: see set_auto_laser_control. Mode As an 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>
	<p>Ctrl, P and Mode are only relevant for Type 3 (= laser power) of the transmitted data word, see read_multi_mcbsp.</p>
Comments	<ul style="list-style-type: none"> Up to V1.6.0, the following applies: Any other already activated McBSP transmission for Online Positioning and any other activated Processing-on-the-fly application with encoder signals or positional values is terminated first. As of V1.6.1, the following applies: Processing-on-the-fly corrections are compatible with the "Fly Extension" Commands. The Processing-on-the-fly corrections of the 3 axes are correspondingly overwritten. They can also be overwritten later by other corrections, for example, encoder-based Processing-on-the-fly corrections. Data types 0...2 are then not used for Processing-on-the-fly corrections. Hence, transmission of extra parameters can be combined even with encoder-based Processing-on-the-fly corrections. However, the McBSP memory transfer cannot be changed. set_multi_mcbsp_in enables inputting by the McBSP interface of up to 8 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 transmission should be occurring when you call set_multi_mcbsp_in. The McBSP interface ignores the first FrameSync signal after a load_program_file or mcbsp_init. That is, data provided is not transmitted, see page 85.

Ctrl Command	set_multi_mcbsp_in
Comments (cont'd)	<ul style="list-style-type: none"> For the transmission, the type assignment must be coded into the 3 least significant bits of the data word according to: <ul style="list-style-type: none"> $\text{McBSPValue} = (\text{Value} \ll 3) \mid \text{Type}$ The RTC6 board stores the data word according to: <ul style="list-style-type: none"> $\text{Memory}[\text{McBSPValue} \& 0x7] = (\text{long} \text{ McBSPValue} \gg 3)$ 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 4 extra parameters are not the same as the free variables, see Chapter 6.9.1 "Free Variables", page 134, 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. If more than 4 McBSP transfers complete within a $10 \mu\text{s}$ clock cycle, then prior values might get overwritten. If the Option Processing-on-the-fly is enabled, then set_multi_mcbsp_in 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. Additionally, laser power can be varied by outputting the transmitted type 3 value at the port assigned by the Ctrl parameter. The initialization value P gets put out immediately. For Mode = 0, subsequent type-3 transfers are outputted directly at the port assigned by Ctrl. For Mode = 1, 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". These laser power variation method can be combined with the "vector-controlled laser control" (with parameterized commands). It cannot be combined with other "Automatic Laser Control" methods. It overwrites other variations sharing the same Ctrl parameter. Though unidentical Ctrl parameters are allowed, they serve no practical purpose. If Ctrl = 0, then laser power variation is switched off. The values transmitted by McBSP continue to be copied into type-sorted memory, but are not put out. Special case: if a "freely definable wobble shape" is active, then P is always regarded (irrespective of the Mode 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 Ctrl parameter has no relevance. It should be set to an invalid value (for example, with Ctrl = 0) to avoid doubled or meaningless output.



Ctrl Command	set_multi_mcbsp_in
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 617, OUT 617: compatibility with "Fly Extension" Commands.
Comments	set_multi_mcbsp_in_list, read_multi_mcbsp, set_wobbel_control, set_wobbel_vector

Normal List Command	set_multi_mcbsp_in_list
Function	Like set_multi_mcbsp_in , but a list command.
Restriction	Like set_multi_mcbsp_in .
Call	<code>set_multi_mcbsp_in_list(Ctrl, P, Mode)</code>
Parameters	<p>Ctrl Like set_multi_mcbsp_in.</p> <p>P Like set_multi_mcbsp_in.</p> <p>Mode Like set_multi_mcbsp_in.</p>
Comments	<ul style="list-style-type: none"> • See set_multi_mcbsp_in.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 617, OUT 617: compatibility with "Fly Extension" Commands.
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 , executes PortOutValue1 and PortOutValue2 assigned to the set_pixel command Number times in immediate succession.
Call	<code>set_n_pixel(PortOutValue1, PortOutValue2, Number)</code>
Parameters	PortOutValue1 Like set_pixel .
	PortOutValue2 Like set_pixel .
	Number Number of pixels. As an unsigned 32-bit value. Allowed value range: [1...($2^{32}-1$)]. 0 is automatically set to 1.
Comments	<ul style="list-style-type: none"> For usage of set_n_pixel, see Chapter 8.7 "Pixel Output Mode", page 262. Before the first set_n_pixel command of a line, set_pixel_line must have been called. set_n_pixel defines the parameters for the following Number (identical) set_pixel commands of an image line. For usage, see comments on set_pixel. If only an individual pixel is to be defined, then set_pixel can be used as an alternative to set_n_pixel. set_pixel is synonymous with set_n_pixel(Number = 1). Outside Pixel Output Mode (if set_n_pixel is not directly preceded by set_pixel_line, set_pixel or set_n_pixel), set_n_pixel is a short list command and otherwise ignored. Under some circumstances, a list_continue might be inserted, see Section "Normal, Short, Variable and Multiple List Commands", page 301.
RTC4→RTC6	New command. For RTC4 Compatibility Mode : see set_pixel .
RTC5→RTC6	Unchanged functionality. See set_pixel .
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_pixel , set_pixel_line , set_pixel_line_3d

Ctrl Command	set_offset								
Function	<p>Defines:</p> <ul style="list-style-type: none"> • The offset for coordinate transformations, see Chapter 8.2 "Coordinate Transformations", page 223. • The global offset in virtual Image field, see Section "Coordinate Transformations in the Virtual Image Field", page 168 								
Call	<code>set_offset(HeadNo, XOffset, YOffset, at_once)</code>								
Parameters	<table> <tr> <td>HeadNo</td> <td>See set_offset_xyz.</td> </tr> <tr> <td>XOffset</td> <td>See set_offset_xyz.</td> </tr> <tr> <td>YOffset</td> <td>See set_offset_xyz.</td> </tr> <tr> <td>at_once</td> <td>See set_offset_xyz.</td> </tr> </table>	HeadNo	See set_offset_xyz .	XOffset	See set_offset_xyz .	YOffset	See set_offset_xyz .	at_once	See set_offset_xyz .
HeadNo	See set_offset_xyz .								
XOffset	See set_offset_xyz .								
YOffset	See set_offset_xyz .								
at_once	See set_offset_xyz .								
Comments	<ul style="list-style-type: none"> • <code>set_offset</code> has the same effect as set_offset_xyz, but leaves <code>ZOffset</code> unchanged. 								
RTC4→RTC6	<p>Unchanged first functionality (offset definition).</p> <p>See set_offset_xyz.</p>								
RTC5→RTC6	<p>Unchanged functionality.</p> <p>See set_offset_xyz.</p>								
Version info	Available as of DLL 600, OUT 600, RBF 600.								
References	set_offset_list , set_offset_xyz , set_angle , set_matrix , set_scale								

Variable List Command	set_offset_list								
Function	Like set_offset_xyz , but a list command.								
Call	<code>set_offset_list(HeadNo, XOffset, YOffset, at_once)</code>								
Parameters	<table> <tr> <td>HeadNo</td> <td>Like set_offset_xyz. However, HeadNo = 4 is not available.</td> </tr> <tr> <td>XOffset</td> <td>Like set_offset_xyz.</td> </tr> <tr> <td>YOffset</td> <td>Like set_offset_xyz.</td> </tr> <tr> <td>at_once</td> <td>Like set_offset_xyz_list.</td> </tr> </table>	HeadNo	Like set_offset_xyz . However, HeadNo = 4 is not available.	XOffset	Like set_offset_xyz .	YOffset	Like set_offset_xyz .	at_once	Like set_offset_xyz_list .
HeadNo	Like set_offset_xyz . However, HeadNo = 4 is not available.								
XOffset	Like set_offset_xyz .								
YOffset	Like set_offset_xyz .								
at_once	Like set_offset_xyz_list .								
RTC4→RTC6	See set_offset_xyz .								
RTC5→RTC6	Unchanged functionality.								
Version info	Available as of DLL 600, OUT 600, RBF 600.								
References	set_offset_xyz								

Ctrl Command	set_offset_xyz
Function	<p>Defines:</p> <ul style="list-style-type: none"> The offset for coordinate transformations, see Chapter 8.2 "Coordinate Transformations", page 223. The global offset in virtual Image field, see Section "Coordinate Transformations in the Virtual Image Field", page 168
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>first</i> scan head connector. = 2: The definition only affects the <i>second</i> scan head connector. = 0, 3: The definition affects <i>both</i> scan head connectors. = 4: The definition affects the virtual Image field. Only the three least significant bits are evaluated.</p> <p>XOffset Offsets for the x direction (coordinate translation relative to the origin of the Cartesian coordinate system). In bits. As a signed 32-bit value. Allowed value range: [-268,435,456...+268,435,455]. Out-of-range values are clipped to the boundary values.</p> <p>YOffset Like XOffset (analogously).</p> <p>ZOffset Offset for the z direction (coordinate translation relative to the origin of the Cartesian coordinate system). In bits. As a signed 32-bit value. Allowed value range: [-524,288...+524,287]. Out-of-range values are clipped to the boundary values.</p> <p>at_once Like set_matrix.</p>
Comments	<ul style="list-style-type: none"> $ZOffset$ is stored only once, applying jointly for both scan head connectors ($HeadNo$ is therefore ignored). $ZOffset \neq 0$ causes a shift of the working plane (opposite to the direction of the laser beam). A positive value increases the z coordinate. For a hypothetical control value of $0 0 0$, this would be by $d = ZOffset / K$ to the plane $z = +d$. On the calibration factor K, see Chapter 7.3.2 "Image Field Size and Image Field Calibration", page 166; furthermore, also 3 and 6 in Chapter 7.3.6 "Output Values to the Scan System", page 180. If $HeadNo = 4$, then $ZOffset$ is ignored. For at_once, see set_matrix.
RTC4→RTC6	<p>New command.</p> <p>In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for $XOffset$, $YOffset$, $ZOffset$ by 16. The allowed value ranges decrease accordingly.</p>
RTC5→RTC6	<p>Unchanged functionality.</p> <p>In RTC4 Compatibility Mode, the RTC6 multiplies the specified value for $ZOffset$ by 16. The allowed value range decreases accordingly.</p>
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_offset_xyz_list , set_offset , set_angle , set_matrix , set_scale , set_defocus



Variable List Command	set_offset_xyz_list
Function	Like set_offset_xyz , but a list command.
Call	<code>set_offset_xyz_list(HeadNo, XOffset, YOffset, ZOffset, at_once)</code>
Parameters	HeadNo Like set_offset_xyz . However, HeadNo = 4 is not available.
	XOffset Like set_offset_xyz .
	YOffset Like set_offset_xyz .
	ZOffset Like set_offset_xyz .
	at_once Like set_matrix_list .
RTC4→RTC6	See set_offset_xyz .
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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>
Parameters	<p>Mask1 16-bit mask. As an 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, see also Chapter 9.3.2 "Conditional Command Execution", page 294: $((IOvalue \text{ AND } Mask1) = Mask1) \text{ AND } (((\text{not } IOvalue) \text{ AND } Mask0) = Mask0)$ (= 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 $10 \mu\text{s}$ clock cycle. The paused list can only be continued 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 conditional pause_list takes precedence over a simultaneously present /STOP signal.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 609, OUT 609, RBF 614.
References	pause_list , restart_list , set_pause_list_not_cond



Ctrl Command	set_pause_list_not_cond
Function	Defines the “NOT” condition at the 16-bit digital input port of the EXTENSION 1 socket connector under which a pause_list automatically is executed.
Call	set_pause_list_not_cond(Mask1, Mask0)
Parameters	<p>Mask1 16-bit mask. As an unsigned 32-bit value. Only the least 16 bits are evaluated.</p> <p>Mask0 Like Mask1.</p>
Comments	<ul style="list-style-type: none"> If a list is currently executed and the following condition is <i>not</i> met conditional commands, see also Chapter 9.3.2 “Conditional Command Execution”, page 294: $((\text{IOvalue AND Mask1}) = \text{Mask1}) \text{ AND } (((\text{not IOvalue}) \text{ AND Mask0}) = \text{Mask0})$ (= 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 $10 \mu\text{s}$ clock cycle. The paused list can only be continued 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_not_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 conditional pause_list takes precedence over a simultaneously present /STOP signal.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 610, OUT 610, RBF 615.
References	pause_list, restart_list, set_pause_list_cond



Variable List Command	set_pixel
Function	In the Pixel Output Mode , defines the laser control parameters (in "Classic Mode" pulse length and analog voltage level) for one pixel in an image line.
Call	set_pixel(PortOutValue1, PortOutValue2)
Parameters	<p>PortOutValue1 For the meaning, see Chapter 8.7.3 "Laser Control", page 264. As an unsigned 32-bit value.</p> <p>PortOutValue2 Like PortOutValue1.</p>
Comments	<ul style="list-style-type: none"> • set_pixel is synonymous with set_n_pixel with Number = 1 (see comments there).
RTC4→RTC6	<ul style="list-style-type: none"> • The RTC6 does <i>not</i> support querying of analog voltage levels (see the RTC4's ADChannel parameter and read_pixel_ad command). • RTC4-Pixel mode 0 is no longer supported. • "Classic Mode" (compatible to RTC4 Pixel Output Mode 1): In RTC4 Compatibility Mode, the RTC6 multiplies the specified value for PulseLength (PortOutValue1) by 8 and the one for AnalogOut (PortOutValue2) by 4. The allowed value ranges decrease accordingly. The other Pixel Output Modes are not RTC4 compatible.
RTC5→RTC6	Unchanged functionality. Only applies to "Classic Mode" , since the other Pixel Output Modes are not supported by the RTC5.
Version info	Last change DLL 604, OUT 604, RBF 609.
References	set_n_pixel , set_pixel_line , set_pixel_line_3d

Normal List Command	set_pixel_line
Function	Activates a Pixel Output Mode and defines various pixel output parameters.
Call	<code>set_pixel_line(Channel, HalfPeriod, dX, dY)</code>
Parameters	<p>Channel Defines the Pixel Output Mode as well as the output ports, where the pixel contents are to be outputted. The pixel contents by themselves are defined in set_pixel or set_n_pixel commands. These must directly follow set_pixel_line. The following applies: <code>Channel = Mode + Port</code>. As an unsigned 32-bit value.</p> <p>Allowed are:</p> <p>Mode = 0: Output of <code>PortOutValue1</code> to the pixel duration (formerly parameter <code>PulseLength</code>) and output of <code>PortOutValue2</code> to the analog output port <code>Port</code>. The galvanometer scanner movement is not continuous, see Section "RTC5→RTC6", page 705. "Classic Mode".</p> <p>Mode = 16: Two outputs at <code>Port</code>. One 32-bit pixel in each output. "Extended Mode".</p> <p>Mode = 32: Two outputs at <code>Port</code>. Two 16-bit pixels in each output. "Fast Mode".</p> <p>Mode = 64: Two outputs at <code>Port</code>. 4 8-bit pixels in each output. "Ultra Fast Mode".</p> <p>Mode = 256: Like Mode = 0. However, the galvanometer scanner movement is continuous, see Section "RTC5→RTC6", page 705.</p> <p>Mode + 512: Like Mode = 0. However, the galvanometer scanner movement with Sky Writing (position-accurate pixel line). Cannot be combined with Mode = 256. See also Chapter 8.7.4 "Synchronization", page 267 and Figure 65.</p> <p>Port = 1: 12-bit analog output port 1 (LASER Connector). See comment, page 704.</p> <p>Port = 2: 12-bit analog output port 2 (LASER Connector. With the RTC6 PCIe Board also the MARKING ON THE FLY socket connector). See comment, page 704.</p> <p>Port = 3: 8-bit digital output port (EXTENSION 2 socket connector).</p> <p>Port = 4: 16-bit digital output port (EXTENSION 1 socket connector).</p> <p>Port = 5: Output of <code>PortOutValue1</code> and <code>PortOutValue2</code> of set_pixel and of set_n_pixel to the pixel duration (formerly parameter <code>PulseLength</code>). Not allowed with Mode = 0 and Mode = 256.</p>

Normal List Command	set_pixel_line	
Parameters (cont'd)	HalfPeriod	<p><i>Half</i> pixel output period. In bits. 1 bit equals $1/64 \mu\text{s}$. As an unsigned 32-bit value. Allowed value range: $[\text{Min} \dots (2^{32}-1)]$ with: Min = 80 for Mode = 0 and 256. Maximum frequency: 0.4 MHz. Min = 40 for Mode = 16. Maximum frequency: 0.8 MHz. Min = 20 for Mode = 32. Maximum frequency: 1.2 MHz. For RTC6 Boards without Option "UFPM" the following applies: Min = 40, maximum frequency: 0.8 MHz. Min = 10 for Mode = 64. Maximum frequency: 3.2 MHz. For RTC6 Boards Option "UFPM" the following applies: Min = 40, maximum frequency: 0.8 MHz.</p>
	dx	Distance in the x direction between adjacent pixels. In bits. As a 64-bit IEEE floating point value.
	dy	Distance in the Y direction between adjacent pixels. In bits. As a 64-bit IEEE floating point value.
Comments	<ul style="list-style-type: none"> Each image line of a pixel image must be started by set_pixel_line. set_pixel_line should be preceded by a Jump command or [*]mark[*] Command to the start point of the image line. Directly after set_pixel_line, the required number of set_pixel and set_n_pixel commands must follow. These transmit the PortOutValue1 and PortOutValue2 parameters. Their meaning depend from Mode and Port (see above, parameter Channel). See also Chapter 8.7.3 "Laser Control", page 264. The first list command after set_pixel_line that is not a set_pixel or set_n_pixel command turns off the Pixel Output Mode. set_pixel_line, too, ends the Pixel Output Mode before starting it again. In the process, a default pixel is inserted. The default pixel for Port 1...4 is defined by set_port_default. But beware: Port numbers of set_port_default and set_pixel_line do not match! The default pixel for Port 5 (PulseLength) is defined by set_default_pixel. With unallowed Channel values (example: 5, for Mode = 0 and Port = 5) set_pixel_line is replaced by a list_nop (get_last_error return code RTC6_PARAM_ERROR). The Pixel Output Mode is not activated in these cases. Note that <i>half</i> the period length must be specified for HalfPeriod. $2 \times \text{HalfPeriod}$ is thus the chronological distance between individual pixels, see Chapter 8.7 "Pixel Output Mode", page 262. HalfPeriod must not be smaller than Min (is automatically clipped). For outputs at the 12-bit analog output port 1 and 12-bit analog output port 2 the following must be obeyed: only for pixel output frequencies up to around 100 kHz (that is, for a HalfPeriod < approx. 320) digital-to-analog conversion is always fully completed. With such pixel output frequencies users must carefully verify whether the results are as expected. With pixel output frequencies > 100 kHz, usage of the UFPM Extension Board, see Chapter 17 "Appendix B: The UFPM Extension Board", page 905, is recommended. 	

Normal List Command	<code>set_pixel_line</code>
Comments (cont'd)	<ul style="list-style-type: none"> The Pixel Output Mode can be combined with Processing-on-the-fly. See also Chapter 8.6 "Processing-on-the-fly", page 241. The Pixel Output Mode <i>should not</i> be used in conjunction with "Automatic Laser Control" (see Chapter 7.4.9 ""Automatic Laser Control"", page 196), if there is a readjustment of the port output, pulse length (<code>PulseLength</code>) or output period (<code>HalfPeriod</code>) of the laser control signals LASER1 and LASER2. The Pixel Output Mode is incompatible with the Softstart Mode (not yet implemented). The Pixel Output Mode <i>cannot</i> be combined with Sky Writing. However, a Sky Writing Mode 1-like movement can be set by Mode + 512. This requires Sky Writing to be switched on. In case of a SCANAhead system, the automatic delay calculation must be switched on in addition. This allows pixel lines to be positioned accurately. This Sky Writing movement cannot be combined with Sky Writing Mode 2 movements. See also: <ul style="list-style-type: none"> Chapter 8.7.4 "Synchronization", page 267 and Figure 65 Chapter 7.2.4 "Sky Writing", page 159 The Pixel Output Mode <i>cannot</i> be combined with Wobbel. See also Chapter 8.4 "Wobbel Mode", page 231. See also Chapter 8.7 "Pixel Output Mode", page 262.
RTC4→RTC6	<ul style="list-style-type: none"> For differences to the Pixel Output Mode of the RTC4, see Section "Special Functions", page 55. In RTC4 Compatibility Mode, the RTC6 multiplies the specified value for <code>HalfPeriod</code> by 8 and those for <code>dx</code> and <code>dy</code> by 16. The allowed value ranges decrease accordingly.
RTC5→RTC6	In "Classic Mode" (RTC5 compatible), frequencies up to 400 kHz are possible. Furthermore, there are additional Pixel Output Modes .
Version info	Last change DLL 604, OUT 604, RBF 609. Additional Pixel Output Modes : 16, 32, 64 and +512.
References	set_default_pixel , set_default_pixel_list , set_pixel , set_n_pixel , set_pixel_line_3d , set_port_default



Multiple List 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	Channel Like set_pixel_line .
	HalfPeriod Like set_pixel_line .
	dX Like set_pixel_line .
	dY Like set_pixel_line .
	dZ Distance in the z direction between adjacent pixels. In bits. As a 64-bit IEEE floating point value.
Comments	<ul style="list-style-type: none"> • set_pixel_line_3d lets you define the pixel spacing (between two adjacent image points on a line) by a 3D vector. • set_pixel_line_3d additionally requires two list-memory positions, if parameter <code>dZ</code> is non-zero. The initial component executes as a short list command before the principal part (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 comments there).
RTC4→RTC6	<p>New command.</p> <p>RTC4 Compatibility Mode: see set_pixel_line.</p> <p>In RTC4 Compatibility Mode, the RTC6 <i>does not</i> multiply the specified value for <code>dZ</code> by 16.</p>
RTC5→RTC6	<p>Basically unchanged functionality.</p> <p>RTC5 Compatibility Mode: see set_pixel_line.</p>
Version info	<p>Available as of DLL 600, OUT 600, RBF 600.</p> <p>Last change: see set_pixel_line.</p>
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 Output port for which a default value is to be defined. As an unsigned 32-bit value. Allowed values: = 0: ANALOG OUT1 output port. See also Section "12-Bit Analog Output Port 1 and 2", page 73. = 1: ANALOG OUT2 output port. See also Section "12-Bit Analog Output Port 1 and 2", page 73. = 2: 8-bit digital output port. See also Section "8-Bit Digital Output Port", page 80. = 3: 16-bit digital output port. See also Section "16-Bit Digital Input Port and 16-Bit Digital Output Port", page 77. = 4: 2-bit digital output port. See also Section "2-Bit Digital Output Port", page 73.</p>
Value	<p>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...65,535]. Higher bits are ignored. For Port = 4: 2-bit values [0...3]. Higher bits are ignored. For Value = “-1” (= $2^{32}-1 = 0xFFFFFFFF$), the corresponding default output functionality is switched off (see comment below).</p>
Comments	<ul style="list-style-type: none"> During initialization of the RTC6 (by load_program_file), the default values of all output ports are set to “-1” (= $2^{32}-1$). If a default value ≠ “-1” has been specified for an output port, it is set to this default value as soon as processing of a list has ended with stop_execution or by an External Stop. 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...3 are also set at the end of Pixel Output Mode, see Chapter 8.7 "Pixel Output Mode", page 262. After initialization of position-dependent 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 control signal parameter Ctrl1), is also outputted when the laser is switched off after marking or when position-dependent 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 Chapter 7.4.1 "Enabling, Activating and Switching Laser Control Signals", page 183 and Section "General Notes", page 197. If the default value is set to “-1”, then the maximum allowed value (4095, 4095, 255 or 65,535) is outputted.



Ctrl Command	set_port_default
Comments (cont'd)	<ul style="list-style-type: none"> If the value for <code>Port</code> is invalid, then <code>set_port_default</code> is not executed (<code>get_last_error</code> return code <code>RTC6_PARAM_ERROR</code>). The default values for the output ports 0...2 can also be defined by <code>set_laser_off_default</code>.
RTC4→RTC6	<p>New command.</p> <p>In RTC4 Compatibility Mode, the RTC6 multiplies Value for <code>Port</code> = 0/1 by 4.</p>
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_laser_off_default , set_default_pixel

Undelayed Short List Command	set_port_default_list				
Function	Like <code>set_port_default</code> , but a list command.				
Call	<code>set_port_default_list(Port, Value)</code>				
Parameters	<table> <tr> <td>Port</td> <td>Like <code>set_port_default</code>.</td> </tr> <tr> <td>Value</td> <td>Like <code>set_port_default</code>.</td> </tr> </table>	Port	Like <code>set_port_default</code> .	Value	Like <code>set_port_default</code> .
Port	Like <code>set_port_default</code> .				
Value	Like <code>set_port_default</code> .				
Comments	<ul style="list-style-type: none"> See <code>set_port_default</code>. 				
RTC4→RTC6	New command.				
RTC5→RTC6	Unchanged functionality.				
Version info	Available as of DLL 609, OUT 609, RBF 614.				
References	set_port_default				

Ctrl Command	set_pulse_picking
Function	Switches on Pulse Picking Laser Mode .
Call	<code>set_pulse_picking(No)</code>
Parameters	<p>No As an unsigned 32-bit value. Allowed value range: [0...63].</p> <p>= 0: LASER2 puts out the LASERON signal.</p> <p>= 1...63: LASER2 puts out each No^{th} LASER1 pulse.</p> <p>$No > 63$: No is clipped to 63 (get_last_error return code <code>RTC6_PARAM_ERROR</code>).</p>
Comments	<ul style="list-style-type: none"> For Pulse Picking Laser Mode, see Chapter 7.4.8 "Pulse Picking Laser Mode", page 195. 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→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_laser_mode , set_pulse_picking_list

Ctrl Command	set_pulse_picking_length
Function	Defines a constant pulse length for the LASER2 signal in Pulse Picking Laser Mode .
Call	<code>set_pulse_picking_length(Length)</code>
Parameters	<p>Length Pulse length. As an unsigned 32-bit value. Allowed value range: [0...65,535]. Higher bits are ignored. 1 bit equals $1/64 \mu s$. The default value after load_program_file is 0.</p>
Comments	<ul style="list-style-type: none"> For the value to take effect, the following must have been activated: <ul style="list-style-type: none"> – Pulse Picking Laser Mode by set_pulse_picking – The “constant pulse length” mode by set_laser_control(Bit #7 = 1). The value then takes immediate effect, even if a marking is carried out. 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 Chapter 7.4.8 "Pulse Picking Laser Mode", page 195. If neither Pulse Picking Laser Mode nor constant pulse length mode has been activated, then the value <code>Length</code> 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→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	–

Undelayed Short List Command	set_pulse_picking_list
Function	Like set_pulse_picking , but a list command.
Call	<code>set_pulse_picking_list(No)</code>
Parameters	No Like set_pulse_picking .
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 53 .
Call	<code>set_qswitch_delay(Delay)</code>
Parameters	Delay Q-Switch delay. As an unsigned 32-bit value. 1 bit equals 1/64 μ s. Allowed value 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→RTC6	New command. In RTC4 Compatibility Mode , the RTC5 multiplies the specified value for <code>Delay</code> by 8. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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	Like set_qswitch_delay , but a list command.
Call	<code>set_qswitch_delay_list(Delay)</code>
Parameters	Delay Like set_qswitch_delay .
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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	<code>set_rot_center(X, Y)</code>
Parameters	X Position of the rotation center referenced to the zero point (0 0) of the Image field . As a signed 32-bit value. Allowed value range: [-2 ²⁴ ...(2 ²⁴ -1)].
	Y Like X (analogously).
Comments	<ul style="list-style-type: none"> For Processing-on-the-fly correction, see Chapter 8.6.3 "Compensating Rotary Movements", page 246. 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 equivalent to 32x the Image field. Usage of a second scan head is only practical if it is set up for exactly the same rotational center.
RTC4→RTC6	Unchanged functionality. In addition: increased value range. In RTC4 Compatibility Mode , the RTC6 multiplies the specified values for X and Y by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_fly_rot , set_fly_rot_pos

Delayed Short List Command	set_rot_center_list
Function	Like set_rot_center , but a list command.
Call	<code>set_rot_center_list(X, Y)</code>
Parameters	X Like set_rot_center .
	Y Like set_rot_center .
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_rot_center

Ctrl Command	set_RTC4_mode
Function	Sets RTC4 Compatibility Mode as the current RTC6 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 RTC6 DLL. It has been made available so that user programs written for the RTC4 can also be processed by the RTC6 (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 RTC6 commands. In the command descriptions of this user manual such changes are noted in the “RTC4→RTC6” row. • RTC6 Standard Mode is predefined as the default RTC6 DLL operation mode and can also be specified subsequently by set_RTC6_mode. • The current RTC6 DLL operation mode can be queried by get_RTC_mode. • set_RTC4_mode is available even without explicit access rights to a particular board. • set_RTC4_mode is not available as a multi-board command. • The scope of set_RTC4_mode is not board-specific, but rather global to the RTC6 DLL and all RTC6 boards to which the user program has access rights. • The board-specific error variables LastError and AccError (see Chapter 6.8 “Error Handling”, page 129) are neither generated nor altered by set_RTC4_mode.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	get_RTC_mode , set_RTC5_mode , set_RTC6_mode

Ctrl Command	set_rtc5_mode
Function	Sets RTC5 Compatibility Mode as the current RTC6 DLL operation mode.
Call	<code>set_rtc5_mode()</code>
Comments	<ul style="list-style-type: none"> • RTC5 Compatibility Mode is an optional operation mode of the RTC6 DLL. It has been made available so that user programs written for the RTC5 can also be processed by the RTC6 (to a large extent) without needing to modify the programming code. However, a prerequisite here is that the program can only contain RTC5 commands that also exist with unchanged functionality as RTC6 commands. In the command descriptions of this user manual such changes are noted in the “RTC5→RTC6” row. See also Chapter 2.12.2 “Adapting RTC5 Source Code for the RTC6 PCIe Board”, step 3, page 60. • In RTC5 Compatibility Mode as RTC6 DLL operation mode Z coordinates and defocus values (for example, parameter <code>Shift</code> of set_defocus) must be specified with a parameter resolution of 16 bits. These values are automatically multiplied by 16. The allowed value ranges decrease accordingly. • In RTC5 Compatibility Mode as RTC6 DLL operation mode the laser delay values (parameter <code>LaserOnDelay</code> and <code>LaserOffDelay</code> of set_laser_delays) must be specified with a parameter resolution of 1/2 μs. These values are automatically multiplied by 32. The allowed value ranges decrease accordingly. • RTC6 Standard Mode is predefined as the default RTC6 DLL operation mode and can also be specified subsequently by set_rtc6_mode. • The current RTC6 DLL operation mode can be queried by get_rtc_mode. • set_rtc5_mode is available even without explicit access rights to a particular RTC6 board. • set_rtc5_mode is not available as a multi-board command. • The scope of set_rtc5_mode is not board-specific, but rather global to the RTC6 DLL and all RTC6 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 129) are neither generated nor altered by set_rtc5_mode.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	get_rtc_mode , set_rtc4_mode , set_rtc6_mode



Ctrl Command	set_RTC6_mode
Function	Sets RTC6 Standard Mode as the current RTC6 DLL operation mode (default setting).
Call	<code>set_RTC6_mode()</code>
Comments	<ul style="list-style-type: none"> In RTC6 Standard Mode as RTC6 DLL operation mode, Z coordinate values and defocus values must be specified with a parameter resolution of 20 bits. In RTC6 Standard Mode as RTC6 DLL operation mode, laser delay values must be specified with a parameter resolution of 1/64 μs. The current RTC6 DLL operation mode can be queried by get_RTC_mode. set_RTC6_mode is available even without explicit access rights to a particular RTC6 board. set_RTC6_mode is not available as a multi-board command. The scope of set_RTC6_mode is not board-specific, but rather global to the RTC6 DLL and therefore, to all RTC6 boards to which the user program has access rights. The board-specific error variables LastError and AccError (see Chapter 6.8 "Error Handling", page 129) are neither generated nor altered by set_RTC6_mode.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	get_RTC_mode , set_RTC4_mode , set_RTC5_mode



Ctrl Command	set_scale
Function	Uses a specified scaling factor common to the x axis and y axis to define the scaling matrix M_S for all subsequent coordinate transformations, see Chapter 8.2 "Coordinate Transformations", page 223 .
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>first</i> scan head connector. = 2: The definition only affects the <i>second</i> scan head connector. = 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 value 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 Determines when the defined transformation becomes effective. Like set_matrix(HeadNo = 0...3). As an unsigned 32-bit value.</p>
Comments	<ul style="list-style-type: none"> See Chapter 8.2 "Coordinate Transformations", page 223.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_scale_list , set_angle , set_matrix , set_offset

Variable List Command	set_scale_list
Function	Like set_scale , but a list command.
Call	<code>set_scale_list(HeadNo, Scale, at_once)</code>
Parameters	<p>HeadNo Like set_scale.</p> <p>Scale Like set_scale.</p> <p>at_once Determines when the defined transformation becomes effective. Like set_matrix_list. As an unsigned 32-bit value.</p>
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_scale



Ctrl Command	set_scanahead_laser_shifts
Comments	<ul style="list-style-type: none"> Only for scan systems with SCANAhead technology, for example, the excelliSCAN. This command is described in the "excelliSCAN Scan Heads – Functional Principle of SCANAhead Servo Control and Operation by RTC6 Boards" Manual.

Undelayed Short List Command	set_scanahead_laser_shifts_list
Comments	<ul style="list-style-type: none"> Only for scan systems with SCANAhead technology, for example, the excelliSCAN. This command is described in the "excelliSCAN Scan Heads – Functional Principle of SCANAhead Servo Control and Operation by RTC6 Boards" Manual.

Ctrl Command	set_scanahead_line_params
Comments	<ul style="list-style-type: none"> Only for scan systems with SCANAhead technology, for example, the excelliSCAN. This command is described in the "excelliSCAN Scan Heads – Functional Principle of SCANAhead Servo Control and Operation by RTC6 Boards" Manual.

Undelayed Short List Command	set_scanahead_line_params_list
Comments	<ul style="list-style-type: none"> Only for scan systems with SCANAhead technology, for example, the excelliSCAN. This command is described in the "excelliSCAN Scan Heads – Functional Principle of SCANAhead Servo Control and Operation by RTC6 Boards" Manual.

Ctrl Command	set_scanahead_params
Comments	<ul style="list-style-type: none"> Only for scan systems with SCANAhead technology, for example, the excelliSCAN. This command is described in the "excelliSCAN Scan Heads – Functional Principle of SCANAhead Servo Control and Operation by RTC6 Boards" Manual.

Ctrl Command	set_scanahead_speed_control
Comments	<ul style="list-style-type: none"> Only for scan systems with SCANAhead technology, for example, the excelliSCAN. This command is described in the "excelliSCAN Scan Heads – Functional Principle of SCANAhead Servo Control and Operation by RTC6 Boards" Manual.

Delayed Short List Command	set_scanner_delays						
Function	Sets the Scanner Delays .						
Call	<code>set_scanner_delays(Jump, Mark, Polygon)</code>						
Parameters	<table> <tr> <td>Jump</td> <td>Scanner delay of type: Jump Delay. As an unsigned 32-bit value. 1 bit equals 10 μs. Allowed value range: [0...(2³²-1)].</td> </tr> <tr> <td>Mark</td> <td>Scanner delay of type: Mark Delay. As an unsigned 32-bit value. 1 bit equals 10 μs. Allowed value range: [0...(2³²-1)].</td> </tr> <tr> <td>Polygon</td> <td>Scanner delay of type: Mark Delay. As an unsigned 32-bit value. 1 bit equals 10 μs. Allowed value range: [0...(2³²-1)].</td> </tr> </table>	Jump	Scanner delay of type: Jump Delay . As an unsigned 32-bit value. 1 bit equals 10 μ s. Allowed value range: [0...(2 ³² -1)].	Mark	Scanner delay of type: Mark Delay . As an unsigned 32-bit value. 1 bit equals 10 μ s. Allowed value range: [0...(2 ³² -1)].	Polygon	Scanner delay of type: Mark Delay . As an unsigned 32-bit value. 1 bit equals 10 μ s. Allowed value range: [0...(2 ³² -1)].
Jump	Scanner delay of type: Jump Delay . As an unsigned 32-bit value. 1 bit equals 10 μ s. Allowed value range: [0...(2 ³² -1)].						
Mark	Scanner delay of type: Mark Delay . As an unsigned 32-bit value. 1 bit equals 10 μ s. Allowed value range: [0...(2 ³² -1)].						
Polygon	Scanner delay of type: Mark Delay . As an unsigned 32-bit value. 1 bit equals 10 μ s. Allowed value range: [0...(2 ³² -1)].						
Comments	<ul style="list-style-type: none"> See also Chapter 7.2.2 "Scanner Delays", page 146. Variable Delays for jumps and Polylines can be set by set_delay_mode. The specified delays are automatically adjusted by the RTC6 to avoid laser control errors, see Section "Automatic Delay Adjustments", page 154. The default setting after load_program_file corresponds to <code>set_scanner_delays(9, 6, 3)</code>. 						
RTC4→RTC6	Unchanged functionality. In addition: increased value range.						
RTC5→RTC6	Unchanged functionality.						
Version info	Available as of DLL 600, OUT 600, RBF 600.						
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 (= 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 value range: [0...(2 ³² -1)].
Comments	<ul style="list-style-type: none"> set_serial is synonymous with set_serial_step with Step = 1 (see comments there).
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 (= 0 after load_program_file).
Call	<code>set_serial_step(No, Step)</code>
Parameters	No Serial number. As an unsigned 32-bit value. Allowed value range: [0...(2 ³² -1)].
	Step Increment size. As an unsigned 32-bit value. Allowed value range: [0...9999]; only the last 4 decimal digits are used.
Comments	<ul style="list-style-type: none"> • load_program_file sets the starting serial number to 0 and the increment size to 1. • If mark_serial or mark_serial_abs has been called with Mode M₂ = 1 (that is, automatic serial-number incrementing has been 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 usage of set_serial_step, see Chapter 7.5.2 "Marking Serial Numbers", page 209.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	mark_serial , mark_serial_abs , set_serial , select_serial_set , select_serial_set_list

Normal List Command	set_serial_step_list
Function	Like set_serial_step , but a list command.
Call	<code>set_serial_step_list(No, Step)</code>
Parameters	No Like set_serial_step .
	Step Like set_serial_step .
Comments	<ul style="list-style-type: none"> • See set_serial_step.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_serial_step , select_serial_set_list , get_list_serial , mark_serial , mark_serial_abs



Ctrl Command	set_sky_writing
Function	Activates Sky Writing Mode 1 and sets the corresponding parameters or switches off Sky Writing .
Call	<code>set_sky_writing(Timelag, LaserOnShift)</code>
Parameters	<p>Timelag Like set_sky_writing_para.</p> <p>LaserOnShift Like set_sky_writing_para.</p>
Comments	<ul style="list-style-type: none"> • set_sky_writing is identical to <code>set_sky_writing_para(Timelag, LaserOnShift, Nprev, Npost)</code> with <code>Nprev</code> = approx. $(0.15 \times \text{Timelag})$ and <code>Npost</code> = approx. $(0.1 \times \text{Timelag})$. • See also information about set_sky_writing_para and Chapter 7.2.4 "Sky Writing", page 159.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality. In RTC5 Compatibility Mode , the RTC6 multiplies the value specified for <code>LaserOnShift</code> by 32. The allowed value range decreases accordingly.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_sky_writing_para , set_sky_writing_list

Ctrl Command	set_sky_writing_limit
Function	Defines the limit for Sky Writing switching in Sky Writing Mode 3 .
Call	<code>set_sky_writing_limit(Limit)</code>
Parameters	<p>Limit Limit value. As a 64-bit IEEE floating point value. Allowed value range: [-1.0...+1.0]. Out-of-range values are clipped to the boundary values.</p>
Comments	<ul style="list-style-type: none"> For usage of set_sky_writing_limit, see Chapter 7.2.4 "Sky Writing", page 159. 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→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_sky_writing_limit_list

Undelayed Short List Command	set_sky_writing_limit_list
Function	Like set_sky_writing_limit , but a list command.
Call	<code>set_sky_writing_limit_list(Limit)</code>
Parameters	Limit Like set_sky_writing_limit .
Comments	<ul style="list-style-type: none"> See set_sky_writing_limit.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_sky_writing_limit



Normal List Command	set_sky_writing_list
Function	Like set_sky_writing , but a list command.
Call	<code>set_sky_writing_list(Timelag, LaserOnShift)</code>
Parameters	Timelag Like set_sky_writing_para . LaserOnShift Like set_sky_writing_para .
Comments	<ul style="list-style-type: none">• See set_sky_writing, set_sky_writing_para and set_sky_writing_para_list.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality. In RTC5 Compatibility Mode , the RTC6 multiplies the value specified for <code>LaserOnShift</code> by 32. The allowed value range decreases accordingly.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_sky_writing , set_sky_writing_para , set_sky_writing_para_list .





Normal List Command	<code>set_sky_writing_mode_list</code>
Function	Like <code>set_sky_writing_mode</code> , but a list command.
Call	<code>set_sky_writing_mode_list(Mode)</code>
Parameters	Mode Like <code>set_sky_writing_mode</code> .
Comments	<ul style="list-style-type: none"> By <code>set_sky_writing_mode_list</code>, Sky Writing Mode 2 and Sky Writing Mode 3 can be activated or deactivated even within a list (switching to or from Mode = 2 or 3). Here, an already-begun Mark command is finished with Sky Writing Mode 1 and the next is started with it. Deactivation of Sky Writing Mode 1 by <code>set_sky_writing_mode_list</code> (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 (for example, a Jump Delay).
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	<code>set_sky_writing_mode</code>

Ctrl Command	set_sky_writing_para																		
Function	Activates Sky Writing Mode 1 and sets the corresponding parameters or switches Sky Writing off.																		
Call	set_sky_writing_para(Timelag, LaserOnShift, Nprev, Npost)																		
Parameters	<p>Timelag Sky Writing parameter. 1.0 equals $1 \mu\text{s}$. The Timelag value is used with an accuracy of $1/64 \mu\text{s}$. As a 64-bit IEEE floating point value.</p> <p>$\geq 1/4$: Sky Writing Mode 1 is activated. From $1/8$ to $1/4$, Sky Writing is activated, but Timelag = 0 is internally applied.</p> <p>$< 1/8$: Sky Writing is deactivated.</p> <p>LaserOnShift Shift (for positive values: delay) of the point of time, where the are switched on (see Figure 46). 1 bit equals $1/64 \mu\text{s}$. As a signed 32-bit value. Negative values smaller than the complete run-in phase are clipped at runtime, therefore the following applies automatically:</p> <table> <tr> <td>LaserOnShift</td> <td>Modus</td> </tr> <tr> <td>$\geq \text{ca. } (-40) \times N_{\text{prev}}$</td> <td>1</td> </tr> <tr> <td>$\geq \text{ca. } (-20) \times N_{\text{prev}}$</td> <td>2 or 3</td> </tr> </table> <p>Nprev Defines the duration of the run-in phase (see Figure 46). 1 bit equals $10 \mu\text{s}$. As an unsigned 32-bit value. Allowed value range: $0 \leq N_{\text{prev}} \leq (2^{32}-1)$.</p> <table> <tr> <td>Run-in duration [μs]</td> <td>Modus</td> </tr> <tr> <td>$20 \times N_{\text{prev}}$</td> <td>1</td> </tr> <tr> <td>$10 \times N_{\text{prev}}$</td> <td>2 or 3</td> </tr> </table> <p>Npost Defines the duration of the run-out phase (see Figure 46). 1 bit equals $10 \mu\text{s}$. As an unsigned 32-bit value. Allowed value range: $0 \leq N_{\text{post}} \leq (2^{32}-1)$.</p> <table> <tr> <td>Run-out duration [μs]</td> <td>Modus</td> </tr> <tr> <td>$20 \times N_{\text{post}}$</td> <td>1</td> </tr> <tr> <td>$10 \times N_{\text{post}}$</td> <td>2 or 3</td> </tr> </table>	LaserOnShift	Modus	$\geq \text{ca. } (-40) \times N_{\text{prev}}$	1	$\geq \text{ca. } (-20) \times N_{\text{prev}}$	2 or 3	Run-in duration [μs]	Modus	$20 \times N_{\text{prev}}$	1	$10 \times N_{\text{prev}}$	2 or 3	Run-out duration [μs]	Modus	$20 \times N_{\text{post}}$	1	$10 \times N_{\text{post}}$	2 or 3
LaserOnShift	Modus																		
$\geq \text{ca. } (-40) \times N_{\text{prev}}$	1																		
$\geq \text{ca. } (-20) \times N_{\text{prev}}$	2 or 3																		
Run-in duration [μs]	Modus																		
$20 \times N_{\text{prev}}$	1																		
$10 \times N_{\text{prev}}$	2 or 3																		
Run-out duration [μs]	Modus																		
$20 \times N_{\text{post}}$	1																		
$10 \times N_{\text{post}}$	2 or 3																		
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 159. If $N_{\text{prev}} \geq 65,535$, then set_sky_writing_para behaves similarly to set_sky_writing: N_{prev} is set to a value of approx. $(0.15 \times \text{Timelag})$. If $N_{\text{post}} \geq 65,535$, then set_sky_writing_para behaves similarly to set_sky_writing: N_{post} is set to a value of approx. $(0.1 \times \text{Timelag})$. set_sky_writing_para cannot be used to switch back and forth between Sky Writing Mode 1, Sky Writing Mode 2 and Sky Writing Mode 3 (the proper command for this is set_sky_writing_mode). 																		

Ctrl Command	set_sky_writing_para
Comments (cont'd)	<ul style="list-style-type: none"> When <code>set_sky_writing_para</code> is called and no list is running or a list has been halted by <code>set_wait</code>, the following applies: <ul style="list-style-type: none"> With <code>Timelag</code> $\geq 1/8$, Sky Writing Mode 1 is activated, if Sky Writing has not been active before. Sky Writing Mode 2 or Sky Writing Mode 3 remain, but the next Mark command is always started in Sky Writing Mode 1 With <code>Timelag</code> $< 1/8$, Sky Writing is deactivated. When <code>set_sky_writing_para</code> is called and a list is running or a list has been halted by <code>pause_list</code>, the following applies: <ul style="list-style-type: none"> If Sky Writing Mode 2 or Sky Writing Mode 3 is active, then <code>set_sky_writing_para</code> is not executed (<code>get_last_error</code> return code <code>RTC6_BUSY</code>) If no Sky Writing or Sky Writing Mode 0 or Sky Writing Mode 1 is active, <code>set_sky_writing_para</code> parameters are going to be effective upon the next list command. <ul style="list-style-type: none"> In Sky Writing Mode 1, an already started Mark command is executed with the previous parameters and ended with Sky Writing Mode 1. In Sky Writing Mode 0 or Sky Writing Mode 1, the next Mark command is started in Sky Writing Mode 1, if <code>Timelag</code> $\geq 1/8$. Otherwise, it is terminated and a Mark Delay is inserted in Sky Writing Mode 1. For Sky Writing Mode 2 or Sky Writing Mode 3, <code>Nprev = 0</code> and/or <code>Npost = 0</code> automatically get(s) internally corrected to 1 (this is not treated as an error). If you subsequently activate Sky Writing Mode 1 (by <code>set_sky_writing_mode</code>), then <code>Nprev = 0</code> and/or <code>Npost = 0</code> is/are reused.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality. In RTC5 Compatibility Mode , the RTC6 multiplies the value specified for <code>LaserOnShift</code> by 32. The allowed value range decreases accordingly.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_sky_writing , set_sky_writing_para_list

Normal List Command	set_sky_writing_para_list
Function	Like set_sky_writing_para , but a list command.
Call	set_sky_writing_para_list(Timelag, LaserOnShift, Nprev, Npost)
Parameters	Timelag Like set_sky_writing_para .
	LaserOnShift Like set_sky_writing_para .
	Nprev Like set_sky_writing_para .
	Npost Like set_sky_writing_para .
Comments	<ul style="list-style-type: none"> • set_sky_writing_para_list can be executed within a list, even if Sky Writing Mode 2 or Sky Writing Mode 3 is active. Here, an already-begun Mark command is finished with Sky Writing Mode 1 and the next is started with it. • By set_sky_writing_para_list, you cannot switch from Sky Writing Mode 2 or Sky Writing Mode 3 to Sky Writing Mode 1 permanently. For this, use set_sky_writing_mode_list. • Deactivation of Sky Writing Mode 1, Sky Writing Mode 2 or Sky Writing Mode 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 (for example, a Jump Delay).
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality. In RTC5 Compatibility Mode , the RTC6 multiplies the value specified for <code>LaserOnShift</code> by 32. The allowed value range decreases accordingly.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_sky_writing_para



Ctrl Command	set_softstart_level
Function	No function.
Call	set_softstart_level(Index, Level)
Parameters	Index As an unsigned 32-bit value. Level As an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> • set_softstart_level has no effect on the user program.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_softstart_level_list

Normal List Command	set_softstart_level_list
Function	No function.
Call	set_softstart_level_list(Index, Level1, Level2, Level3)
Parameters	Index Like set_softstart_level . Level1 Like set_softstart_level . Level2 Like set_softstart_level . Level3 Like set_softstart_level .
Comments	<ul style="list-style-type: none"> • See set_softstart_level.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_softstart_level



Ctrl Command	set_softstart_mode
Function	No function.
Call	<code>set_softstart_mode(Mode, Number, Delay)</code>
Parameters	<p>Mode As an unsigned 32-bit value.</p> <p>Number As an unsigned 32-bit value.</p> <p>Delay As an unsigned 32-bit value.</p>
Comments	<ul style="list-style-type: none"> • set_softstart_mode has no effect on the user program.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	–

Variable List Command	set_softstart_mode_list
Function	No function.
Call	<code>set_softstart_mode_list(Mode, Number, Delay)</code>
Parameters	<p>Mode Like set_softstart_mode.</p> <p>Number Like set_softstart_mode.</p> <p>Delay Like set_softstart_mode.</p>
Comments	<ul style="list-style-type: none"> • See set_softstart_mode.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	–

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 Signals for "Laser Active" Operation and "laser standby" operation.
Call	<code>set_standby(HalfPeriod, PulseLength)</code>
Parameters	<p>HalfPeriod <i>Half of the standby output period.</i> As an unsigned 32-bit value. 1 bit equals 1/64 μs. Allowed value range: [0...+(2³²-1)].</p> <p>PulseLength <i>Pulse length of the standby pulses.</i> As an unsigned 32-bit value. 1 bit equals 1/64 μs. Allowed value range: [0...(2³²-1)].</p>
	<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 Chapter 7.4 "Laser Control", page 183. At the same time, the signal level of the standby pulses are set by set_laser_control. The standby pulses are available in <i>all</i> laser modes (CO₂ Mode, YAG Mode 1, YAG Mode 2, YAG Mode 3, Laser Mode 4, YAG Mode 5, Laser Mode 6). The standby pulses can be deactivated (turned off) by setting the standby pulse length and/or the standby output period to zero (default). With HalfPeriod, <i>half</i> the standby output period must be specified, see Figure 52 and Figure 54. If PulseLength is larger than the output period (2 \times HalfPeriod), the laser is permanently on. The laser mode is set with set_laser_mode, see also Chapter 7.4 "Laser Control", page 183. To set the active output period and pulse length for the "laser active" laser control signals (beside for Laser Mode 4 and Laser Mode 6), are set by set_laser_pulses_ctrl, set_laser_pulses or set_laser_timing.
RTC4→RTC6	Basically unchanged functionality. In addition: increased value range. In RTC4 Compatibility Mode , the RTC6 multiplies the specified values for HalfPeriod and PulseLength by 8. The allowed value ranges decrease accordingly.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_standby_list , get_standby



Delayed Short List Command	set_standby_list
Function	Like set_standby , but a list command.
Call	<code>set_standby_list(HalfPeriod, PulseLength)</code>
Parameters	HalfPeriod Like set_standby .
	PulseLength Like set_standby .
RTC4→RTC6	See set_standby .
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_standby

Ctrl Command	set_start_list
Function	Opens the list memory for writing of list commands and sets the input pointer to the start of the specified list ("List 1" or "List 2").
Call	<code>set_start_list(ListNo)</code>
Parameters	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"> • <code>set_start_list</code> is synonymous with <code>set_start_list_pos</code> for <code>Pos = 0</code>. • Alternatively, <code>set_start_list_1</code> and <code>set_start_list_2</code> (with no parameter) can be used.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_start_list

Ctrl Command	set_start_list_2
Function	See set_start_list .
Call	<code>set_start_list_2()</code>
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_start_list

Ctrl Command	set_start_list_pos				
Function	Opens the list memory for writing of list commands and sets the input pointer to the specified <i>relative</i> position in the specified list ("List 1" or "List 2").				
Call	<code>set_start_list_pos(ListNo, Pos)</code>				
Parameters	<table> <tr> <td>ListNo</td> <td>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"].</td> </tr> <tr> <td>Pos</td> <td>Position of the input pointer (offset relative to the start of the respective list). As an unsigned 32-bit value. Allowed value range: [0...(2²³-1)].</td> </tr> </table>	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 ²³ -1)].
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 ²³ -1)].				
Comments	<ul style="list-style-type: none"> The next list command is stored at the specified address and all further list commands at the subsequent addresses in the selected list. The specified 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 104 and load_list. The input pointer <i>cannot</i> be set to the protected list memory 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>config_list(Mem1, -1)</code>. After writing, configuration of the area's protection should be restored: <code>config_list(Mem1, Mem2)</code>. For uneven ListNo 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 list status set, READY list status reset, USED list status reset. The LOAD list status of the other corresponding list is reset. <code>set_start_list_pos</code> triggers a flush of the buffered list input, see Chapter 6.4.1 "Loading Lists", page 104. <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. CAUTION: If the end of the respective list area is reached, the list input pointer is automatically reset to the start of the same list area. Make sure not to overwrite any commands still needed by your user program. 				
RTC4→RTC6	New command.				
RTC5→RTC6	Unchanged functionality.				
Version info	Available as of DLL 600, OUT 600, RBF 600.				
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 value range: [0...1023]).
	Pos Absolute start address. As an unsigned 32-bit value. Allowed value range: [0...(2 ²³ -1)].
Comments	<ul style="list-style-type: none"> If <code>Index > 1023</code> and/or <code>Pos > (2²³-1)</code>, then set_sub_pointer is not executed (get_last_error return code <code>RTC6_PARAM_ERROR</code>). The set_sub_pointer command can be used for referencing a nonindexed subroutine, which thereby becomes an indexed subroutine that is protectable by save_disk/load_disk and/or callable by the index. set_sub_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_sub_pointer can be queried by get_input_pointer before loading the command lists. set_sub_pointer only stores <i>starting addresses</i> in the internal management table. An indexed subroutine 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 has been finalized by list_return and does not contain set_end_of_list.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 value range: [0...41]. The same assignment applies as for load_text_table: = 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 value range: [0...(2²³-1)].</p>
Comments	<ul style="list-style-type: none"> Indexed text strings can be used for marking time, date or serial numbers, see Section "Calling Indexed Text Strings", page 119. If <code>Index > 41</code> and/or <code>Pos > (2²³-1)</code>, then set_text_table_pointer is not executed (get_last_error return code <code>RTC6_PARAM_ERROR</code>). set_text_table_pointer 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 has been finalized by list_return and does not contain set_end_of_list.
RTC4→RTC6	<p>New command.</p> <p>set_text_table_pointer is synonymous with set_char_table, which is available for the RTC4 SCANalone Board (standalone version of the RTC4 board).</p>
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	load_text_table , mark_date , mark_serial , mark_time



Ctrl Command	set_timelag_compensation	
Function	Compensates the various Tracking errors of the xy axes and z axis.	
Call	set_timelag_compensation(HeadNoXY, TimelagXY, TimelagZ)	
Parameters	HeadNoXY	Number of the xy scan head connector. As an unsigned 32-bit value. Allowed value range: [1...2].
	TimelagXY	Tracking error of the xy axis in [10 μ s]. As an unsigned 32-bit value. Allowed value range: [0...255].
	TimelagZ	Tracking error of the z axis in [10 μ s]. As an unsigned 32-bit value. Allowed value range: [0...255].
Comments	<ul style="list-style-type: none"> Unallowed parameter values produce a get_last_error return code of RTC6_PARAM_ERROR and set_timelag_compensation is not executed. If list execution is currently active, then the get_last_error return code gets set to RTC6_BUSY (the control command get_status returns BUSY list execution status or INTERNAL-BUSY list execution status or PAUSED list execution status) and set_timelag_compensation is not executed. Faster axes are held back towards the slower axes. set_timelag_compensation can also be used with scan heads of the excelliSCAN series. If set_scanahead_params configured the RTC6 board for controlling an excelliSCAN, then TimelagXY is ignored. Instead, the PreviewTime parameter value from set_scanahead_params gets applied. set_timelag_compensation automatically waits until a HEAD_BUSY gets reset (see "excelliSCAN Scan Heads – Functional Principle of SCANAhead Servo Control and Operation by RTC6 Boards" Manual), hence execution of set_timelag_compensation can last up to PreviewTime. With SCANAhead systems, the parameter PreviewTime from set_scanahead_params is automatically used as "Tracking error". 	
RTC5→RTC6	New command.	
RTC5→RTC6	New command.	
Version info	Available as of DLL 600, OUT 600, RBF 600.	
References	set_scanahead_params , get_last_error , get_status	

Delayed Short List Command	set_trigger
Function	Starts measurement value recording of the specified signals.
Call	<code>set_trigger(Period, Signal1, Signal2)</code>
Parameters	<p>Period Measurement period (time interval between 2 data pair recordings). As an unsigned 32-bit value. 1 bit equals 10 μs. Allowed value range: [0...(2³¹-1)]. Bit #31 = 0: Automatic stop at the max. channel size, see set_trigger4. Bit #31 = 1: Endless recording (circular buffer).</p> <p>Signal1 Signal type for measurement channel 1. As an unsigned 32-bit value. Allowed value range: see below.</p> <p>0 LASERON signal. 1 = laser control signal on, 0 = laser control signal off.</p> <p>1 StatusAX. Status signal of x axis, first scan head connector.</p> <p>2 StatusAY. Status signal of y axis, first scan head connector.</p> <p>3 Reserved.</p> <p>4 StatusBX. Status signal of x axis, second scan head connector.</p> <p>5 StatusBY. Status signal of y axis, second scan head connector.</p> <p>6 Reserved.</p> <p>7 SampleX (up to 4 in 7.3.6, page 180). Cartesian control value for the x axis.</p> <p>8 SampleY (up to 4 in 7.3.6, page 180). Cartesian control value for the y axis.</p> <p>9 SampleZ (up to 4 in 7.3.6, page 180). Cartesian control value for the z axis.</p> <p>10 SampleAX_Corr (up to 8 in 7.3.6, page 180). Corrected x axis control value for the first scan head connector.</p> <p>11 SampleAY_Corr (up to 8 in 7.3.6, page 180). Corrected y axis control value for the first scan head connector.</p> <p>12 SampleAZ_Corr (up to 8 in 7.3.6, page 180). Corrected z axis control value, if xy are connected to the first scan head connector. Identical to the effective output value for the z axis.</p>

Delayed Short List Command	set_trigger		
Parameters (cont'd)	Signal1 (cont'd)	13	SampleBX_Corr (up to 8 in 7.3.6, page 180). Corrected control value for the x axis, second scan head connector.
		14	SampleBY_Corr (up to 8 in 7.3.6, page 180). Corrected control value for the y axis, second scan head connector.
		15	SampleBZ_Corr (up to 8 in 7.3.6, page 180). Corrected z axis control value, if xy are connected to the second scan head connector. Identical to the effective output value for the z axis.
		16	StatusAX+LASERON. StatusAX for laser control signal on, -524,288 for laser control signal off.
		17	StatusAY+LASERON. StatusAY for laser control signal on, -524,288 for laser control signal off.
		18	StatusBX+LASERON. StatusBX for laser control signal on, -524,288 for laser control signal off.
		19	StatusBY+LASERON. StatusBY for laser control signal on, -524,288 for laser control signal off.
		20	SampleAX_Out (up to 9 in 7.3.6, page 180). Actual output value for the x axis, first scan head connector. Not usable for z axis output values.
		21	SampleAY_Out (up to 9 in 7.3.6, page 180). Actual output value for the y axis, first scan head connector. Not usable for z axis output values.
		22	SampleBX_Out (up to 9 in 7.3.6, page 180). Actual output value for the for the x axis, second scan head connector. Not usable for z axis output values.
		23	SampleBY_Out (up to 9 in 7.3.6, page 180). Actual output value for the y axis, second scan head connector. Not usable for measuring z axis output values.
		24	Laser control parameter of "Automatic Laser Control". See set_auto_laser_control .

Delayed Short List Command	set_trigger		
Parameters (cont'd)	Signal1 (cont'd)	25	SampleAX_Trans (up to 7 in 7.3.6, page 180). Transformed control value for the x axis, first scan head connector.
		26	SampleAY_Trans (up to 7 in 7.3.6, page 180). Transformed control value for the y axis, first scan head connector.
		27	SampleAZ_Trans (up to 7 in 7.3.6, page 180). Transformed z axis control value, if xy are connected to the first scan head connector.
		28	SampleBX_Trans (up to 7 in 7.3.6, page 180). Transformed control value for the x axis, second scan head connector.
		29	SampleBY_Trans (up to 7 in 7.3.6, page 180). Transformed control value for the y axis, second scan head connector.
		30	SampleBZ_Trans (up to 7 in 7.3.6, page 180). Transformed z axis control value, if xy are connected to the second scan head connector.
		31	Laser control parameter of "vector-controlled laser control". See set_vector_control .
		32	Focus shift. See set_vector_control , set_defocus , set_defocus_list .
		33	12-bit output value at the ANALOG OUT1 output port. See set_vector_control and Chapter 9.1.4 "12-Bit Analog Output Port 1 and 2" , page 282.
		34	12-bit output value at the ANALOG OUT2 output port. See set_vector_control and Chapter 9.1.4 "12-Bit Analog Output Port 1 and 2" , page 282.
		35	Output value at the 16-bit digital output port. See set_vector_control and Chapter 9.1.1 "16-Bit Digital Output Port" , page 281.
		36	Output value at the 8-bit digital output port. See set_vector_control and Chapter 9.1.2 "8-Bit Digital Output Port" , page 282.
		37	Pulse length (PulseLength) of the laser control signals LASER1 and LASER2. See set_vector_control .
		38	Output period (HalfPeriod) of the laser control signals LASER1 and LASER2. See set_vector_control .



Delayed Short List Command	set_trigger
Parameters (cont'd)	<p>Signal1 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 port (EXTENSION 1).</p> <p>47 Only for intelliWELD II: Zoom value.</p> <p>48 FreeVariable4.</p> <p>49 FreeVariable5.</p> <p>50 FreeVariable6.</p> <p>51 FreeVariable7.</p> <p>52 32-bit "Timestamp Counter". See Chapter 8.12 "Time Measurements", page 280.</p> <p>53 Wobble amplitude. See set_wobble, set_wobble_mode and Chapter 8.4 "Wobble Mode", page 231.</p> <p>54 ReadAnalogIn. See read_analog_in.</p> <p>55 Scaled encoder value for X.</p> <p>56 Scaled encoder value for Y.</p> <p>57 Scaled encoder value for Z.</p> <p>58 RS-232.</p>



Delayed Short List Command	set_trigger
Parameters (cont'd)	Signal1 59 <code>read_mcbsp(0)</code> (cont'd) 60 <code>read_multi_mcbsp(0)</code> 61 <code>read_multi_mcbsp(1)</code> 62 <code>read_multi_mcbsp(2)</code> 63 Reserved. 64 Reserved. 65 Reserved. 66 Reserved.
Signal2	Like (analogously) Signal1.

Delayed Short List Command	set_trigger
Comments	<p><i>General Comments</i></p> <ul style="list-style-type: none"> • If Signal1 and Signal2 are not from the above list, then set_trigger (except for <code>Period = 0</code>) is replaced with a list_nop (get_last_error return code <code>RTC6_PARAM_ERROR</code>). • After a set_trigger with <code>Period > 0</code>, a data pair is recorded immediately and subsequently at intervals defined by the specified measurement period. <ul style="list-style-type: none"> – <code>Period</code> (Bit #31 = 0): <ul style="list-style-type: none"> • The measurement value recording ends automatically with 2^{24} data pairs (if Signal2 is invalid, even with 2^{25} data entries, see also set_trigger4). • It ends earlier, if one of the following occurs: <ul style="list-style-type: none"> - set_trigger (<code>Period = 0</code>) is called - set_trigger4 (<code>Period = 0</code>) is called – <code>Period</code> (Bit #31 = 1): <ul style="list-style-type: none"> • Starts an endless data recording according <code>Period = "measurement period 0x80000000"</code> with $0 < \text{measurement period} < 2^{31}-1$. In this case, the data recording does not end automatically. Instead, it starts from the beginning each time and overwrites previously recorded data (ring buffer). In the meantime, the data can be read out with get_waveform_offset from any location and in any packets. • measurement_status also provides information about the current status of the measurement value recording. • A measurement value recording started by set_trigger occurs only during the execution of a list. • Given the measurement value recording has not been terminated by set_trigger (<code>Period = 0</code>) or set_trigger4 (<code>Period = 0</code>), it is automatically continued with the start of a new list (the status of the measurement value recording is not reset by set_end_of_list). During the execution of a list, the status is reset by set_trigger (<code>Period = 0</code>), set_trigger4 (<code>Period = 0</code>) or stop_execution. By stop_trigger the status can be reset if no list is executed. • Sample values and status values returned by the scan system and stored on the RTC6 by set_trigger (<code>Signal1, Signal2 = 1...23, 25...30</code>) are always in the RTC6 20-bit range, even if the RTC6 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. It is generally recommended to end the measurement explicitly before reading out the data. The measured values are transferred to the PC by get_waveform as 32-bit values and must be evaluated accordingly by users (see comments for get_value). • The current status of the measurement value recording can be queried by measurement_status. • For aborting a measurement value recording by set_trigger (<code>Period = 0</code>) or set_trigger4 (<code>Period = 0</code>), the values of Signal1 and Signal2 are irrelevant.

Delayed Short List Command	set_trigger
Comments (cont'd)	<ul style="list-style-type: none"> 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 counter halts at its most recent value. This allows subsequent querying by measurement_status of the number of data entries. In contrast, if a measurement session is newly started with set_trigger (<code>Period > 0</code>) or set_trigger4(<code>Period > 0</code>), the measurement counter is reset and the measurements obtained thus far are overwritten. It is not possible to resume an explicitly or automatically halted measurement session. <p><i>Comments on the Data</i></p> <ul style="list-style-type: none"> 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. control_command can be used with iDRIVE scan systems (see Glossary entry on page 26) to specify which information is returned on the status channels. Only X measurement signals contain meaningful data with single-status channel scan systems. 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 second scan head connector's X channel (select_cor_table(<code>HeadA, 0</code>)), then its status signal can be queried by StatusBX (<code>Signal1/Signal2 = 4</code>). 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>. Status signals <code>Status<...></code> are a few 10 µs clock cycles later than control signals <code>Sample<...></code>. See also Section "Reading Out Data", page 212. <code>Signal1, Signal2 = 0, 24, 31...42 and 47...51</code> enables logging of values that were outputted during the previous clock cycle. Data recording of different signals is not synchronous because of the internal processing chain. Depending on how the signals are combined, you must later correct the data by a fixed time offset. <code>Signal1, Signal2 = 24</code> enables logging of the signal parameter that is outputted by "Automatic Laser Control" (but not with vector-defined laser control; see set_auto_laser_control). Logging only occurs when "Automatic Laser Control" has been activated and the laser is on. When the laser is switched off, 0 is recorded. With "Spot Distance Control", signal 24 has no direct meaning, it does <i>not</i> mean "HalfPeriod" in particular. <code>Signal1, Signal2 = 31</code> enables logging of the signal parameter specified for "vector-controlled laser control" (see set_vector_control). Logging is also possible without executing [*]para[*] Commands, but then the signal values remain unchanged.

Delayed Short List Command	set_trigger
Comments (cont'd)	<ul style="list-style-type: none"> Pulse length and output period (Signal1, Signal2 = 37, 38) are only logged for Signals for "Laser Active" Operation (not for standby signals). In Softstart Mode (not yet implemented) and in Pixel Output Mode, the pulse length and output period might possibly change faster than the 10 μs clock cycle (time resolution of logging by set_trigger). These values cannot be recorded. For Signal1, Signal2 = 39...42 and 48...51, see Chapter 6.9.1 "Free Variables", page 134.
RTC4→RTC6	Basically unchanged functionality. However: <ul style="list-style-type: none"> The measurement session can be aborted by Period = 0. The signals StatusAZ and StatusBZ (Signal1/Signal2 = 3, 6) are no longer available. All coordinate values are referred to (0 0 0) (value range [-2¹⁹...(2¹⁹-1)]) and are no longer zero point shifted.
RTC5→RTC6	Basically unchanged functionality. Increased memory area for data recordings by merging channels, see set_trigger4 .
Version info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 610, OUT 610, RBF 615: endless recording.
References	get_waveform, get_waveform_offset, 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 the measurement value recording of the specified measurement signals (has the same effect as set_trigger , but with up to 4 simultaneous measurement signals).
Call	<code>set_trigger4(Period, Signal1, Signal2, Signal3, Signal4)</code>
Parameters	Period Like set_trigger .
	Signal1 Like set_trigger .
	Signal2 Like set_trigger .
	Signal3 Like set_trigger .
	Signal4 Like set_trigger .
Comments	<ul style="list-style-type: none"> See comments for set_trigger. 2^{23} entries per measurement channel are available with set_trigger4, if all 4 signals are within the allowed range (see set_trigger). The measurement value recording ends automatically with 2^{23} data <i>quadruplets</i>. If Signal 3 and Signal 4 are not allowed (see set_trigger), set_trigger4 is synonymous with set_trigger. The memory areas of Signal 3 and Signal 4 are added to the memory areas of Signal 1 and Signal 2. This allows recording of 2^{24} data pairs. The measured value recording ends automatically with 2^{24} data <i>pairs</i>. If only Signal 1 is within the permissible range, the memory areas of all 4 channels are merged into a single one. With this up to 2^{25} data values can be recorded. The measured value recording ends automatically at 2^{25} data <i>values</i>. set_trigger(<code>Period = 0</code>) and set_trigger4(<code>Period = 0</code>) both terminate active measurement value recording regardless of which command has started it. If set_trigger(<code>Period > 0</code>) or set_trigger4(<code>Period > 0</code>) is used to start a new measurement value recording, then the measurement value counter is reset. Existing data is overwritten with new measurement values. See also the corresponding comments for set_trigger. As with set_trigger, get_waveform can be used to transfer recorded values to the PC.
RTC4→RTC6	New command.
RTC5→RTC6	Basically unchanged functionality. Increased memory area for data recordings by merging unused channels, endless recording.
Version info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 610, OUT 610, RBF 615: see set_trigger .
References	set_trigger , stop_trigger , get_waveform , get_waveform_offset

Undelayed Short List Command	set_vector_control	
Function	Initializes or deactivates “vector-controlled laser control”. See Section “Vector-Defined Laser Control”, page 205 .	
Call	set_vector_control(<i>Ctrl</i> , <i>Value</i>)	
Parameters	<i>Ctrl</i>	<p>Control parameter for initializing or deactivating “vector-controlled laser control” (for <i>Ctrl</i> = 1...6: identical with set_auto_laser_control).</p> <p>As an unsigned 32-bit value.</p> <p>= 1...8: Defines which signal parameter is to be varied by “vector-controlled 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 (PulseLength) of the laser control signals LASER1 and LASER2. = 5: Output period (HalfPeriod) of the laser control signals LASER1 and LASER2. = 6: Output value at the 16-bit digital output. = 7: Focus shift (“Defocus”). = 8: Reserved for intelliWELD. <p>= 0 or > 8: deactivates “vector-controlled laser control”.</p>
	<i>Value</i>	<p>Defines the starting value for the parameter selected by <i>Ctrl</i>.</p> <p>As an unsigned 32-bit value.</p> <p>Allowed values (out-of-range values are clipped to the boundary values):</p> <ul style="list-style-type: none"> For <i>Ctrl</i> = 1/2: 12-bit values [0...4095]. For <i>Ctrl</i> = 3: 8-bit values [0...255]. For <i>Ctrl</i> = 4: [0...(2³²-1)]. 1 bit equals 1/64 μs. For <i>Ctrl</i> = 5: [0...(2³²-1)]. 1 bit equals 1/64 μs. For <i>Ctrl</i> = 6: 16-bit values [0...65,535]. For <i>Ctrl</i> = 7: [-524,288...+524,287], Value = focus shift + 524,288.
Comments	<ul style="list-style-type: none"> • If <i>Ctrl</i> is an invalid value, then “vector-controlled laser control” is deactivated (<i>Ctrl</i> = 0, initialization state). Otherwise, <i>Value</i> is applied as the starting value of the next [*]para[*] Command. If <i>Value</i> is invalid, then it is clipped to the maximum allowed value. • set_vector_control only affects [*]para[*] Commands. • If an “Automatic Laser Control” has been activated by set_auto_laser_control with the same control parameter (<i>Ctrl</i> = 1...6), then set_vector_control sets the 100% value of the “Automatic Laser Control” to value <i>Value</i>. However, this does not apply to <i>Ctrl</i> = 5: set_auto_laser_control(<i>Ctrl</i> = 7) (“Spot Distance Control”) and “vector-controlled laser control” with set_vector_control (<i>Ctrl</i> = 5) (HalfPeriod) <i>cannot</i> be combined. 	

Undelayed Short List Command	set_vector_control
Comments (cont'd)	<ul style="list-style-type: none"> For <code>Ctrl = 7</code>, the focus shift is linearly varied as with set_defocus and set_defocus_list, see comments there) with [*]para[*] Commands (in order for the z outputs to be outputted, the Option "3D" must be enabled and a 3D correction file must be loaded and assigned as well) up to the specified end value (this requires enabling the Option "3D" as well as loading and assigning a 3D correction file). If, for the first [*]para[*] Command, the currently set focus shift does not match the initial value (<code>Value</code>) specified by set_vector_control, then the command begins with a "Hard jump" (in the z output) to <code>Value</code>. The setting defined by set_defocus or set_defocus_list is lost. Unlike signed values in the range $[-524,288 \dots +524,287]$ for set_defocus and set_defocus_list, the focus shift (<code>Value</code>) specified for set_vector_control must be an unsigned number shifted upward by 524,288: <code>Value</code> = focus shift + 524,288. "Automatic Laser Control" should not be combined with the variable laser power of set_multi_mcbsp_in or the "freely definable wobble shape".
RTC4→RTC6	<p>New command.</p> <p>This command has no RTC4 Compatibility Mode for <code>Value</code>.</p> <p>In RTC4 Compatibility Mode, the RTC6 multiplies with <code>Ctrl = 7</code> the specified value for <code>Value</code> by 16 (<code>Value</code> = focus shift + 32,768).</p> <p>The allowed value range decreases accordingly.</p>
RTC5→RTC6	<p>Unchanged functionality.</p> <p>In RTC5 Compatibility Mode, the RTC6 multiplies with <code>Ctrl = 7</code> the specified value for <code>Value</code> by 16 (<code>Value</code> = focus shift + 32,768).</p> <p>The allowed value range decreases accordingly.</p>
Version info	<p>Available as of DLL 600, OUT 600, RBF 600.</p> <p>Last change DLL 634: <code>Ctrl = 8</code>.</p>
References	set_auto_laser_control



Ctrl Command	set_verify
Function	Activates or deactivates a download verification. See Chapter 6.8.1 "Download Verification", page 130 .
Call	OldVerify = set_verify(Verify)
Parameters	Verify Setting parameter. As an unsigned 32-bit value. = 0: Verification is deactivated. > 0: Verification is activated.
Result	The Verify setting parameter that has been active before calling set_verify . As an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> If verification is activated, the download times are extended. Verification of correction file downloads only works if the correction file contains a checksum (otherwise the get_last_error return code RTC6_VERIFY_ERROR 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 does not contain a checksum, then verify_checksum enters it. set_verify is available even without explicit access rights to a specific RTC6 board. set_verify only changes settings in the RTC6 DLL of the specified (or default) board. It has no effect on the board itself. The board-specific error variables LastError and AccError (see Chapter 6.8 "Error Handling", page 129) are neither generated nor altered by set_verify. Activation of the download verification by set_verify can generate the get_last_error return code RTC6_OUT_OF_MEMORY, if a RTC6 DLL-internal Windows memory request fails. In this case, the download verification remains deactivated.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	get_error , get_last_error , verify_checksum

Normal List Command	set_wait
Function	Inserts a numbered break point ("wait marker") into the list.
Call	<code>set_wait(WaitWord)</code>
Parameters	<p>WaitWord Number of the break point. As an unsigned 32-bit value. Allowed value range: [1...(2³²-1)]. 0 is corrected to 1.</p>
Comments	<ul style="list-style-type: none"> The list processing is interrupted at each break point and remains halted until continued by <code>release_wait</code>. This provides a way to implement synchronizations. The Signals for "Laser Active" Operation are switched off by <code>set_wait</code> and a home jump defined by <code>home_position</code> or <code>home_position_xyz</code> might be executed (the INTERNAL-BUSY list execution status is set while the home jump is executed). Continuation by <code>execute_list_pos</code>, <code>restart_list</code> or an External Start is consequently not possible. However, <code>stop_execution</code> or an External Stop is possible. <code>release_wait</code>, <code>stop_execution</code> or an External Stop removes suppression of the start. <code>set_wait</code> sets the PAUSED list execution status and resets the BUSY list execution status (both queryable with <code>get_status</code>). The opposite occurs with a subsequent <code>release_wait</code>, see also Chapter 6.4.3 "List Execution Status", page 107. If processing has been stopped at a break point, then <code>get_wait_status</code> returns the number of this break point.
RTC4→RTC6	Basically unchanged functionality. However: Additional PAUSED list execution status which is set by <code>set_wait</code> .
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	<code>get_wait_status</code> , <code>release_wait</code> , <code>pause_list</code> , <code>stop_list</code>

Delayed Short List Command	set_wobbel
Function	Defines the parameters for an ellipse-shaped wobbel motion. "Wobbel" is a motion of the output position which is added to the regular marking movement. See Chapter 8.4 "Wobbel Mode", page 231 .
Call	<code>set_wobbel(Transversal, Longitudinal, Freq)</code>
Parameters	<p>Transversal Amplitude of the elliptical <i>perpendicular</i> to the momentary direction of motion or to the one defined by set_wobbel_direction. In bits. As an unsigned 32-bit value. Allowed value range: [0...131.071 (=2^{17}-1)]. Larger values are clipped.</p> <p>Longitudinal Amplitude of the elliptical <i>parallel</i> to the momentary direction of motion or to the one defined by set_wobbel_direction. In bits. As an unsigned 32-bit value. Allowed value range: [0...131.071 (=2^{17}-1)]. Larger values are clipped.</p> <p>Freq Frequency of the wobbel movement in Hz (number of ellipses per second). As a 64-bit IEEE floating point value. Allowed value range: [-6000...6000]. Larger values are clipped.</p>
	<ul style="list-style-type: none"> The Wobbel 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 wobbel 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 mark speed should be coordinated, see Chapter 8.4.1 "Wobbel Shapes – Important Notes on Choosing Appropriate Parameter Values", page 233). For arcs, too, the wobbel motion follows the current marking direction (exception: see below). Therefore, independently of the Cartesian angle, the effective line width is always the same.
	<ul style="list-style-type: none"> The Wobbel 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 wobbel vector rotates clockwise for positive values and counterclockwise 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 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).
Comments	<ul style="list-style-type: none"> The Wobbel 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 wobbel 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 mark speed should be coordinated, see Chapter 8.4.1 "Wobbel Shapes – Important Notes on Choosing Appropriate Parameter Values", page 233). For arcs, too, the wobbel motion follows the current marking direction (exception: see below). Therefore, independently of the Cartesian angle, the effective line width is always the same. The Wobbel 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 wobbel vector rotates clockwise for positive values and counterclockwise 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 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).

Delayed Short List Command	set_wobbel
Comments (cont'd)	<ul style="list-style-type: none"> For identical amplitudes (Longitudinal = Transversal), the wobbel startpoint is permanently referenced to the coordinate system, that is, independent of the current direction of motion. By set_wobbel_direction, a fixed reference direction can be set which differs from it. 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. 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", page 233. The present wobbel amplitude can be recorded by set_trigger/set_trigger4 (signal 53). The format of the data is ((transversal << 16) + longitudinal).
RTC4→RTC6	<p>Basically unchanged functionality.</p> <p>More wobbel shape: elliptical, direction dependent adjustable, see also set_wobbel_mode.</p> <p>In RTC4 Compatibility Mode, the RTC6 multiplies the specified value for Longitudinal and Transversal by 16. The allowed value ranges decrease accordingly.</p>
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_mark_speed, set_wobbel_mode, set_wobbel_direction

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>
Parameters	<p>Ctrl Control parameter for initializing or deactivating laser power variation. As an unsigned 32-bit value.</p> <ul style="list-style-type: none"> = 0: Deactivates laser power variation. = 1...6: Defines which signal parameter to vary. On the meaning, see set_auto_laser_control. = 7: Deactivates laser power variation (get_last_error return code RTC6_PARAM_ERROR). = 8: Activates laser power variation where ANALOG OUT1 and ANALOG OUT2 alternate. > 8: Same as 7. <p>Value Nominal laser power P0 (100%). As an unsigned 32-bit value. Allowed value range: see set_auto_laser_control; the maximum is [0...65,535] or 0xFFFFFFFF (with Ctrl = 1...6) or 0xFFFF (with Ctrl = 8). Excessive values are clipped.</p> <p>MinValue Limit that cannot be exceeded, see set_auto_laser_control. As an unsigned 32-bit value. The allowed value range depends on the selected Ctrl parameter and on Value.</p> <p>.MaxValue See MinValue.</p>
Comments	<ul style="list-style-type: none"> • Any still-pending delayed short list command executes first. • For Ctrl = 0, Ctrl = 7 and Ctrl > 8, laser power variation is switched off (initialization state after load_program_file). The values for McBSP multi transfers are then not used. • Ctrl = 8 activates a laser power variation where ANALOG OUT1 and ANALOG OUT2 alternate. The “Free definable wobbel figure” starts with ANALOG OUT1. Subsequently, alternation occurs with every Microstep. See also set_wobbel_vector_2. • With Ctrl = 8, Value, MinValue and MaxValue are to be interpreted as 2 independent 16-bit values: <ul style="list-style-type: none"> – The upper 16 bits apply to ANALOG OUT2 – The lower 16 bits apply to ANALOG OUT1 • set_wobbel_control(Ctrl = 8) and set_wobbel_mode(Mode = 3), see page 760, cannot be combined.



Undelayed Short List Command	set_wobbel_control
Comments (cont'd)	<ul style="list-style-type: none"> The conditions for Value, MinValue, MaxValue are the same as for "Automatic Laser Control" (see set_auto_laser_control), but with a maximum of [0...65,535]. Initialized values are MinValue = 0 and MaxValue = 0xFFFFFFFF, that is, no restrictions. The laser power can be combined with the "vector-controlled laser control", if the corresponding Ctrl parameters have been chosen identically. If you want variable laser power along a wobbel shape, then set_wobbel_control must execute before you activate the "freely definable wobbel shape" by set_wobbel_mode. Otherwise, laser power is not varied, even if you make a change in the data sets. Special case: For <ul style="list-style-type: none"> Value = 0xFFFFFFFF (with Ctrl = 1...6) Value = 0xFFFF (with Ctrl = 8) the nominal laser power is derived from the port assigned by the signal parameter Ctrl, instead of from set_wobbel_control. This is then the current content at this timepoint set by other normal commands, for example, write_da_1_list for Ctrl = 1. These are executed prior to 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→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 631, OUT 632: With Ctrl = 8, now alternation occurs with every Microstep (no longer with every set_wobbel_vector call).
References	set_wobbel_vector , set_multi_mcbsp_in , set_multi_mcbsp_in_list , set_wobbel_vector_2



Undelayed Short List Command	set_wobbel_direction
Function	Defines a direction vector for wobbel motions.
Call	<code>set_wobbel_direction(dx, dy)</code>
Parameters	<code>dx</code> x component of the direction vector. In bits. As a signed 32-bit value.
	<code>dy</code> y component of the direction vector. In bits. As a signed 32-bit value.
Comments	<ul style="list-style-type: none"> For non-zero direction vectors (<code>dx</code> and/or <code>dy</code> 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 RTC6 normalizes the direction vector. If <code>dx = dy = 0</code>, 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→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_wobbel, set_wobbel_direction, set_wobbel_mode, set_wobbel_offset

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 Chapter 8.4 "Wobbel Mode", page 231 .
Call	<code>set_wobbel_mode(Transversal, Longitudinal, Freq, Mode)</code>
Parameters	<p>Transversal Amplitude of the wobbel movement <i>perpendicular</i> to the momentary direction of motion or to the one defined by set_wobbel_direction. In bits. As an unsigned 32-bit value. Allowed value range: [0...131.071 (=2^{17}-1)]. Excessive values are clipped.</p> <p>Longitudinal Amplitude of the wobbel movement <i>parallel</i> to the momentary direction of motion or to the one defined by set_wobbel_direction. In bits. As an unsigned 32-bit value. Allowed value 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). As a 64-bit IEEE floating point value. Allowed value range: [-6000...6000]. Larger values are clipped.</p> <p>Mode Defines the wobbel shape. As a signed 32-bit value. = 0: Ellipse-shaped wobbel movement. < 0: Figure-of-8 wobbel movement perpendicular to the motion direction (vertical 8). = 1: Figure-of-8 wobbel movement parallel to the motion direction (horizontal 8). > 1: "Freely definable wobbel shape", see set_wobbel_vector.</p>
Comments	<ul style="list-style-type: none"> If <code>Mode = 0</code>, set_wobbel_mode functions identically to set_wobbel (see comments there). If <code>Mode ≠ 0</code>, then a figure-of-8 wobbel movement 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 set_wobbel_mode executes while a list contains a "freely definable wobbel shape" (see set_wobbel_vector and Chapter 8.4 "Wobbel Mode", page 231), then <code>Mode > 1</code> selects this shape, otherwise the horizontal figure-8 shape is selected similarly to <code>Mode = 1</code>. If you <i>subsequently</i> define a wobbel shape and the Wobbel Mode has been activated with the parameter <code>Mode > 1</code>, then a switch from the horizontal figure-8 to the wobbel shape automatically occurs. But beware: it is then not possible to vary the power along the wobbel figure, see set_wobbel_control.

Delayed Short List Command	set_wobbel_mode
Comments (cont'd)	<ul style="list-style-type: none"> The Wobbel 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. During the figure-of-8's first loop, the wobbel vector rotates clockwise for positive values and counterclockwise 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 Polyline (including arcs). For identical amplitudes (Longitudinal = Transversal), the wobbel startpoint is permanently referenced to the coordinate system, that is, independent of the current direction of motion. For an angle, elliptical or figure-8 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 and is used upon the next switch-on of the Wobbel Mode with Mode > 1. This also applies, if the wobbel figure has been previously saved by create_dat_file and then loaded by load_program_file. 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", page 233. The present wobbel amplitude can be recorded by set_trigger/set_trigger4 (signal 53). 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→RTC6	<p>New command.</p> <p>In RTC4 Compatibility Mode, the RTC6 multiplies the specified value for Longitudinal and Transversal by 16. The allowed value ranges decrease accordingly.</p>
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_wobbel, set_wobbel_control, set_wobbel_offset, set_wobbel_vector



Delayed Short List Command	set_wobbel_mode_phase
Function	Switches a classical Wobbel Mode on and off as with set_wobbel_mode . In addition, defines a start phase.
Call	<code>set_wobbel_mode_phase(Transversal, Longitudinal, Freq, Mode, Phase)</code>
Parameters	<p>Transversal As with set_wobbel_mode.</p> <p>Longitudinal As with set_wobbel_mode.</p> <p>Freq As with set_wobbel_mode.</p> <p>Mode As with set_wobbel_mode.</p> <p>Phase Start phase. As a 64-bit IEEE floating point value. Allowed value range: [0.0°...360.0°]. Negative values are clipped to 0.0°. Too big positive values are reduced by the corresponding multiples of 360.0°.</p>
Comments	<ul style="list-style-type: none"> • Mode ≥ 2 ("freely definable wobbel shape") is not allowed with set_wobbel_mode_phase (is clipped to 1). • The start phase is converted to an integer value with a 16-bit resolution for a full circle. • By set_wobbel_mode_phase, the wobbel shape continues also with: <ul style="list-style-type: none"> – jump commands – timed_jump commands – para_jump commands – list_nop – long_delay Here, the wobbel shape follows the last valid marking direction. • set_wobbel_direction and set_wobbel_offset are effective.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 609, OUT 609, RBF 614.
References	set_wobbel_mode

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	<code>set_wobbel_offset(OffsetTrans, OffsetLong)</code>
Parameters	<p>OffsetTrans Transversal offset. As a signed 32-bit value. Allowed value range: $\pm 32,767$. Larger values are clipped. Initialization values after load_program_file: (0,0).</p> <p>OffsetLong Longitudinal offset. Otherwise, like OffsetTrans.</p>
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 231. 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→RTC6	New command. In RTC4 Compatibility Mode the RTC6 multiplies the specified values for OffsetTrans and OffsetLong by 16. The allowed value ranges decrease accordingly.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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>
Parameters	<p>dTrans Microstep of a linear wobbel shape section. In bits. dLong As a 64-bit IEEE floating point value. Period Allowed value range: [-256.0...+255.0]. dTrans is the wobbel excursion perpendicular to the direction of motion (which is either the laser trajectory (see Glossary entry on page 29) or the direction of motion defined by set_wobbel_direction). dLong is correspondingly longitudinal to it.</p> <p>Period As an unsigned 32-bit value. Period Allowed value range: [0...65,535]. = 1...65,535: number of Microsteps. = 0: The wobbel shape is switched off.</p> <p>dPower Microstep of the relative laser power. dPower As a 64-bit IEEE floating point value. dPower Allowed value range: [-1.0...+1.0].</p> <p>Out-of-range values are clipped to the boundary values.</p>
Comments	<ul style="list-style-type: none"> 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 set_wobbel_vector begin a new wobbel shape. Each call of set_wobbel_vector adds a new section at the end of the previous section independently of when the call is performed. Up to 1023 wobbel shape sections can be defined. After 1023 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), that is, directly on the marking itself (the “Hard jump” of “classic” wobbel shapes is eliminated if the longitudinal amplitude $\neq 0$) 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 Microstep is executed within $10\ \mu\text{s}$. 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 get executed as a “Hard jump”. With the last step the laser power is reset to the initial nominal laser power. The summed up wobbel amplitude may never exceed $\pm(2^{17}-1)$.

Undelayed Short List Command	set_wobbel_vector
Comments (cont'd)	<ul style="list-style-type: none"> If activated by set_wobbel_control and set_wobbel_mode(Mode = 2), the following applies: <ul style="list-style-type: none"> The RTC6 varies the current laser power P within the wobbel shape section according to $P(i, n) = P_{\text{nom}} \times (1 + (\sum_{j=1}^{i-1} d\text{Power}_j \times \text{Period}_j) + d\text{Power}_i \times n)$ with $1 \leq i \leq$ wobbel vector number and $1 \leq n \leq \text{Period}_i$. n represents the current number of each Microstep. Nominal laser power P_{nom} is set by the list command set_wobbel_control. If activated by set_multi_mcbsp_in or set_multi_mcbsp_in_list, the nominal laser power P_{nom} is multiplicatively varied by the laser power parameter P across multiple McBSP transfers: $P_{\text{nom McBSP}} = P_{\text{nom}} \times P / 16,384$. Beware here that for each both corrections above a maximum laser power of only $P_{\text{nom}} \times 4.0$ 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 $\text{Ctrl} = 5$ (half period), $d\text{Power}$ 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 P_{nom} 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 P_{nom} is emitted at the port assigned by Ctrl if set_wobbel_control has been last called with $\text{Value} = 0xFFFFFFFF$. Otherwise, the laser power P_{nom} multiplied by the external factor from set_multi_mcbsp_in is emitted.

Undelayed Short List Command	set_wobbel_vector <ul style="list-style-type: none"> • If activated by set_wobbel_control and set_wobbel_mode(Mode = 3), the following applies: <ul style="list-style-type: none"> – The RTC6 varies the current laser power P within the wobbel shape section according to $P(i, n) = P_{100} \times (\text{Factor} + (\sum_{j=1}^{i-1} d\text{Power}_j \times \text{Period}_j) + d\text{Power}_i \times n)$ – P_{100} (as with Mode = 2) is the nominal laser power as it would be without the wobbel figure. It is set by set_wobbel_control(Ctrl, Value, MinValue, MaxValue): <ul style="list-style-type: none"> • With Value = 0...65,535, the specified Value value is used • With Value = 0xFFFFFFFF, the value is taken over that has been last outputted to the port Ctrl. – Factor is an unsigned 16+bit value with scaling 16,384 = one (default value). The initial Factor value can be set to nnnn by Value = 0xFF00nnnn and another set_wobbel_control call. The parameters Ctrl, MinValue and MaxValue are taken over from the latest set_wobbel_control call. – To change the 100% performance, set_wobbel_control can be called at any time. However, the initial Factor value only becomes effective when the Wobbel Mode is switched on <i>newly</i>. At wobbel shape runtime, Factor cannot be changed by set_wobbel_control. – Factor is changed per Microstep additively by the dPower value from set_wobbel_vector(dTrans, dLong, Period, dPower). The maximum value is 65,535 (Factor 4) and cannot be exceeded. – Power modulation via McBSP (as with Mode = 2) is not possible.
---	--



Undelayed Short List Command	set_wobbel_vector
Comments (cont'd)	<ul style="list-style-type: none"> 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 Microsteps 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", page 233. create_dat_file saves a "freely definable wobbel shape", see comment on page 348.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_multi_mcbsp_in , set_multi_mcbsp_in_list , set_wobbel_control , set_wobbel_offset , set_wobbel_vector_2



Undelayed Short List Command	set_wobbel_vector_2
Function	Like set_wobbel_vector . However, only for set_wobbel_control (<code>Ctrl</code> = 8).
Call	<code>set_wobbel_vector_2(dTrans, dLong, Period, dPower, dPower2, Ctrl)</code>
Parameters	<p><code>dTrans</code> Like set_wobbel_vector.</p> <p><code>dLong</code> Like set_wobbel_vector.</p> <p><code>Period</code> Like set_wobbel_vector.</p> <p><code>dPower</code> Microstep of the relative laser power. Refers to ANALOG OUT1. As a 64-bit IEEE floating point value. Allowed value range: [-1.0...+1.0].</p> <p><code>dPower2</code> Microstep of the relative laser power. Refers to ANALOG OUT2. As a 64-bit IEEE floating point value. Allowed value range: [-1.0...+1.0].</p> <p><code>Ctrl</code> Reserved. As an unsigned 32-bit value. Out-of-range values are clipped to the boundary values.</p>
Comments	<ul style="list-style-type: none"> • <code>set_wobbel_vector_2</code> can only be used with set_wobbel_control(<code>Ctrl</code> = 8). • Like set_wobbel_vector.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 631 , OUT 632 .
References	set_wobbel_vector

Ctrl Command	set_zoom
Call	<code>set_zoom(Zoom)</code>
Comments	<ul style="list-style-type: none"> • This command is described, for example, in the manual for the intelliWELD II FT.

Undelayed Short List Command	set_zoom_list
Call	<code>set_zoom_list(Zoom)</code>
Comments	<ul style="list-style-type: none"> • This command is described, for example, in the manual for the intelliWELD II FT.



Ctrl Command	simulate_encoder
Function	Activates or deactivates encoder simulation for the specified encoder.
Call	<code>simulate_encoder(EncoderNo)</code>
Parameters	<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 Section "Encoder Simulation", page 298). • 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>RTC6_PARAM_ERROR</code>).
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 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 value 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 Starts, see Section "External Start", page 289. The track delay is specified in <i>relative</i> counting units of the selected encoder counter (the RTC6 encoder counters are triggered by an external or simulated encoder signal, see Chapter 9.3.3 "Synchronization by Encoder Signals", page 297). The start trigger for the start occurs only after the internal encoder counter has reached the specified track delay. External Starts initiated by simulate_ext_start or by an external start signal, but whose execution has been postponed according to the specified track delay, are placed in a queue that accommodates up to 8 starts. simulate_ext_start cancels a previous queue and starts a new one. A start trigger initiated by simulate_ext_start or an external start signal only triggers a start if it does not occur when the BUSY list execution status is set (for example, when outputting a list), when the INTERNAL-BUSY list execution status is set (for example, during goto_xy) and/or when the PAUSED list execution status is set (after pause_list, stop_list or set_wait). Otherwise, Bit #11 of the get_startstop_info return value is set. Therefore, if an unsuitable track delay is specified (for example, <code>Delay = 0</code>), no start is triggered. If simulate_ext_start is the first command in a list, then get_startstop_info 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 workpiece's direction of motion). Track delays can also be set with set_ext_start_delay or set_ext_start_delay_list. Track delays are deactivated by initialization (with load_program_file), by external stops and by stop_execution. They can also be deactivated by set_control_mode (Bit #2). The simulate_ext_start 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, simulate_ext_start_ctrl 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, set_fly_2d, set_fly_2d or set_fly_rot. If <code>EncoderNo > 1</code>, then simulate_ext_start is replaced with a list_nop (get_last_error return code RTC6_PARAM_ERROR).

Normal List Command	<code>simulate_ext_start</code>
RTC4→RTC6	Unchanged functionality. In addition: increased value range. In RTC4 Compatibility Mode , the RTC6 multiplies the specified value for <code>Delay</code> by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	<code>simulate_ext_start_ctrl</code> , <code>set_ext_start_delay</code> , <code>set_ext_start_delay_list</code> , <code>set_extstartpos</code> , <code>set_extstartpos_list</code> , <code>set_control_mode</code> , <code>simulate_ext_stop</code>

Ctrl Command	<code>simulate_ext_start_ctrl</code>
Function	Causes a simulated External Start .
Call	<code>simulate_ext_start_ctrl()</code>
Comments	<ul style="list-style-type: none"> For External Starts, see Section "External Start", page 289. The start trigger for the start occurs only after a track delay previously set by <code>set_ext_start_delay</code>, <code>set_ext_start_delay_list</code> or <code>simulate_ext_start</code>. Track delays are deactivated by initialization (with <code>load_program_file</code>), by external stops and by <code>stop_execution</code>. They can also be deactivated with <code>set_control_mode</code> (Bit #2). External Starts initiated by <code>simulate_ext_start_ctrl</code> or by an external start signal, but whose execution has been postponed according to the specified track delay, are placed in a queue that accommodates up to 8 starts. In contrast to <code>simulate_ext_start</code>, <code>simulate_ext_start_ctrl</code> does <i>not</i> cancel the previous queue. A start trigger initiated by <code>simulate_ext_start_ctrl</code> or by an external start signal does only trigger a start if it does not coincide with the output of a list (otherwise, Bit #11 of the <code>get_startstop_info</code> return value gets set). The <code>simulate_ext_start_ctrl</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_ctrl</code>, <code>simulate_ext_start</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>, <code>set_fly_2d</code> or <code>set_fly_rot</code>. <code>simulate_ext_start_ctrl</code> can be disabled by <code>set_control_mode</code> (Bit #4 = 1) or <code>set_control_mode_list</code> (Bit #4 = 1). As of version DLL 609, OUT 609, RBF 614, the following applies: provided that a start is allowed, <code>simulate_ext_start_ctrl</code> waits 30 µs before it returns. This closes a potential timing gap (with <code>get_status</code>) between command call and actual start.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 609, OUT 609, RBF 614: see above.
References	<code>simulate_ext_start</code>



Ctrl Command	simulate_ext_stop
Function	Causes a simulated External Stop .
Call	<code>simulate_ext_stop()</code>
Comments	<ul style="list-style-type: none">For external stops, see Section "External Stop", page 288.simulate_ext_stop simultaneously halts the master board and all slave boards.In contrast, stop_execution only halts the master board.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	simulate_ext_start , simulate_ext_start_ctrl , stop_execution , sync_slaves



Undelayed Short List Command	spot_distance
Function	As spot_distance_ctrl , but an undelayed short list command.
Call	<code>spot_distance(Dist)</code>
Parameters	Dist See spot_distance_ctrl .
Comments	<ul style="list-style-type: none"> • See spot_distance_ctrl.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 609, OUT 609, RBF 613.
References	spot_distance_ctrl , set_auto_laser_control , set_auto_laser_params , set_auto_laser_params_list

Ctrl Command	spot_distance_ctrl
Function	Defines the geometric pulse distance for the "Automatic Laser Control" with <code>Ctrl = 7</code> .
Call	<code>spot_distance_ctrl(Dist)</code>
Parameters	Dist Pulse distance. In bits. As a 64-bit IEEE floating point value.
Comments	<ul style="list-style-type: none"> • The pulse distance is resolved with an accuracy of 1/40 bit (Image field coordinates). • <code>Dist = 0</code> suppresses the "Automatic Laser Control" with <code>Ctrl = 7</code>, but does not switch it off. • <code>Dist</code> must not exceed the maximum value 26,214. A practical <code>Dist</code> value depends on mark speed, laser frequency and scan system.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 609, OUT 609, RBF 613.
References	spot_distance , set_auto_laser_control , set_auto_laser_params , set_auto_laser_params_list

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 execution restart activated with <code>start_loop</code> repeats until execution is ended by calling <code>quit_loop</code>. <code>start_loop</code> can be called at any time but only has effect upon the next <code>set_end_of_list</code>. If the automatic list change is activated during processing of a list, then upon reaching <code>set_end_of_list</code> execution continues without delay at the other list. If there is only one list (<code>Mem2 = 0</code>, see <code>config_list</code>), then upon reaching <code>set_end_of_list</code> execution continues at Pos = 0 (that is, 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 "Changing Lists Automatically", page 109. So that <code>start_loop</code> can function at all, the already active list must absolutely be finalized by <code>set_end_of_list</code>; 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 <code>set_end_of_list</code>, 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, <code>start_loop</code> and <code>auto_change_pos(Pos > 0)</code> are called, then upon the next <code>set_end_of_list</code> the command <code>auto_change_pos(Pos > 0)</code> is executed; and at the next one <code>start_loop</code> is executed. The current <code>List Status</code> can be queried by <code>read_status</code>. The current <code>List Execution Status</code> can be queried by <code>get_status</code>. <code>start_loop</code> triggers a flush of the buffered list input, see Chapter 6.4.1 "Loading Lists", page 104.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	quit_loop

Ctrl Command	stepper_abs						
Function	Triggers set-position movements to the specified absolute set positions by both stepper motor output ports.						
Call	<code>stepper_abs(Pos1, Pos2, WaitTime)</code>						
Parameters	<table> <tr> <td>Pos1</td> <td>Absolute set position in CLOCK pulse units for stepper motor output ports 1. As signed 32-bit values. Allowed value range: $[-2^{31} \dots + (2^{31}-1)]$.</td> </tr> <tr> <td>Pos2</td> <td>Like Pos1 (analogously).</td> </tr> <tr> <td>WaitTime</td> <td>Determines when stepper_abs returns at the latest. As an unsigned 32-bit value. <i>1 bit equals 1 s.</i> Allowed value range: $[0 \dots + (2^{32}-1)]$.</td> </tr> </table>	Pos1	Absolute set position in CLOCK pulse units for stepper motor output ports 1. As signed 32-bit values. Allowed value range: $[-2^{31} \dots + (2^{31}-1)]$.	Pos2	Like Pos1 (analogously).	WaitTime	Determines when stepper_abs returns at the latest. As an unsigned 32-bit value. <i>1 bit equals 1 s.</i> Allowed value range: $[0 \dots + (2^{32}-1)]$.
Pos1	Absolute set position in CLOCK pulse units for stepper motor output ports 1. As signed 32-bit values. Allowed value range: $[-2^{31} \dots + (2^{31}-1)]$.						
Pos2	Like Pos1 (analogously).						
WaitTime	Determines when stepper_abs returns at the latest. As an unsigned 32-bit value. <i>1 bit equals 1 s.</i> Allowed value range: $[0 \dots + (2^{32}-1)]$.						
Comments	<ul style="list-style-type: none"> For programming the stepper motor signals, see Chapter 9.1.5 "Controlling Stepper Motors", page 283. 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 Section "Notes", page 283. 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), stepper_abs does not execute (get_last_error return code RTC6_PARAM_ERROR). If the CLOCK pulse period has been 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 stepper_abs returns immediately so that system control is restored to the user program. 						
RTC4→RTC6	New command.						
RTC5→RTC6	Unchanged functionality.						
Version info	Available as of DLL 600, OUT 600, RBF 600.						
References	stepper_abs_no , stepper_rel , stepper_rel_no , stepper_abs_list						



Undelayed Short List Command	stepper_abs_list				
Function	Like stepper_abs , but a list command and without WaitTime parameter.				
Call	<code>stepper_abs_list(Pos1, Pos2)</code>				
Parameters	<table> <tr> <td>Pos1</td> <td>Like stepper_abs.</td> </tr> <tr> <td>Pos2</td> <td>Like stepper_abs.</td> </tr> </table>	Pos1	Like stepper_abs .	Pos2	Like stepper_abs .
Pos1	Like stepper_abs .				
Pos2	Like stepper_abs .				
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→RTC6	New command.				
RTC5→RTC6	Unchanged functionality.				
Version info	Available as of DLL 600, OUT 600, RBF 600.				
References	stepper_abs				

Ctrl Command	stepper_abs_no						
Function	Triggers a set-position movement to the specified absolute set position at <i>one</i> stepper motor output port.						
Call	<code>stepper_abs_no(No, Pos, WaitTime)</code>						
Parameters	<table> <tr> <td>No</td> <td>Number of the stepper motor output port. As an unsigned 32-bit value. Allowed values: = 1: Stepper motor output port 1. = 2: Stepper motor output port 2. If the value is invalid, then stepper_abs_no is not executed (get_last_error return code RTC6_PARAM_ERROR).</td> </tr> <tr> <td>Pos</td> <td>Absolute set position in CLOCK pulse units. As a signed 32-bit value. Allowed value range: $[-2^{31} \dots + (2^{31}-1)]$.</td> </tr> <tr> <td>WaitTime</td> <td>Determines when stepper_abs_no returns at the latest. <i>1 bit equals 1 s.</i> As an unsigned 32-bit value. Allowed value range: $[0 \dots + (2^{32}-1)]$.</td> </tr> </table>	No	Number of the stepper motor output port. As an unsigned 32-bit value. Allowed values: = 1: Stepper motor output port 1. = 2: Stepper motor output port 2. If the value is invalid, then stepper_abs_no is not executed (get_last_error return code RTC6_PARAM_ERROR).	Pos	Absolute set position in CLOCK pulse units. As a signed 32-bit value. Allowed value range: $[-2^{31} \dots + (2^{31}-1)]$.	WaitTime	Determines when stepper_abs_no returns at the latest. <i>1 bit equals 1 s.</i> As an unsigned 32-bit value. Allowed value range: $[0 \dots + (2^{32}-1)]$.
No	Number of the stepper motor output port. As an unsigned 32-bit value. Allowed values: = 1: Stepper motor output port 1. = 2: Stepper motor output port 2. If the value is invalid, then stepper_abs_no is not executed (get_last_error return code RTC6_PARAM_ERROR).						
Pos	Absolute set position in CLOCK pulse units. As a signed 32-bit value. Allowed value range: $[-2^{31} \dots + (2^{31}-1)]$.						
WaitTime	Determines when stepper_abs_no returns at the latest. <i>1 bit equals 1 s.</i> As an unsigned 32-bit value. Allowed value range: $[0 \dots + (2^{32}-1)]$.						
Comments	<ul style="list-style-type: none"> A set-position movement is only performed at the stepper motor output port specified by No. Otherwise, stepper_abs_no is identical to stepper_abs (see comments there). 						
RTC4→RTC6	New command.						
RTC5→RTC6	Unchanged functionality.						
Version info	Available as of DLL 600, OUT 600, RBF 600.						
References	stepper_abs , stepper_abs_no_list						



Undelayed Short List Command	stepper_abs_no_list
Function	Like 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 stepper_abs_no_list is, already during loading, replaced by a list_nop (get_last_error return code RTC6_PARAM_ERROR).</p> <p>Pos Like 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→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 period of the CLOCK signals for stepper motor output ports 1. 1 bit equals $10 \mu\text{s}$. As a signed 32-bit value. Allowed values: $[0\dots+(2^{24}-1)]$ 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> <p>Period2 Pulse period of the CLOCK signals for stepper motor output ports 2. Otherwise, like Period1.</p>
Comments	<ul style="list-style-type: none"> For programming the stepper motor signals, see Chapter 9.1.5 "Controlling Stepper Motors", page 283. Period1 or Period2 = 0 can be used as an emergency stop, see also the Section "Terminating Infinite Movements", page 285.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	stepper_control_list

Undelayed Short List Command	stepper_control_list
Function	Like stepper_control , but a list command.
Call	<code>stepper_control_list(Period1, Period2)</code>
Parameters	<p>Period1 Like stepper_control.</p> <p>Period2 Like stepper_control.</p>
Comments	<ul style="list-style-type: none"> See stepper_control.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 port 1 are to be used. As a signed 32-bit value.. Allowed value range: $[-2^{32} \dots + (2^{32}-1)]$.</p> <p>> 0: The SWITCH signal at the stepper motor input port is not used. = 0: The SWITCH signal at the stepper motor input port is used. < 0: The use of the stepper motor input signal remains unchanged.</p> <p>Disable2 Like Disable1 (analogously).</p>
Comments	<ul style="list-style-type: none"> For programming the stepper motor signals, see Chapter 9.1.5 "Controlling Stepper Motors", page 283. The limit switch can be ignored during normal forwarding motions. For example, this may make sense for continuously rotating axes. The SWITCH signals are always used with forwarding motions initiated by stepper_init.
Version info	Available as of DLL 542, OUT 542.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	stepper_init

Ctrl Command	stepper_enable
Function	Sets the ENABLE signals of the stepper motor control.
Call	<code>stepper_enable(Enable1, Enable2)</code>
Parameters	<p>Enable1 ENABLE signal for stepper motor output 1. As a signed 32-bit value. Allowed value range: $[-2^{32} \dots + (2^{32}-1)]$. > 0: The ENABLE signal at the stepper motor output gets set. = 0: The ENABLE signal at the stepper motor output gets reset. < 0: The stepper motor output signal remains unchanged.</p> <p>Enable2 Like <code>Enable1</code> (analogously).</p>
Comments	<ul style="list-style-type: none"> For programming the stepper motor signals, see Chapter 9.1.5 "Controlling Stepper Motors", page 283.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	stepper_enable_list

Undelayed Short List Command	stepper_enable_list
Function	Like stepper_enable , but a list command.
Call	<code>stepper_enable_list(Enable1, Enable2)</code>
Parameters	<p>Enable1 Like stepper_enable.</p> <p>Enable2 Like stepper_enable.</p>
Comments	<ul style="list-style-type: none"> See stepper_enable.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	stepper_enable

Ctrl Command	stepper_init
Function	Performs a stepper motor initialization.
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 stepper_init is not executed (get_last_error return code RTC6_PARAM_ERROR).</p> <p>Period Pulse period of the CLOCK signal. As an unsigned 32-bit value. 1 bit equals 10 μs. Allowed value 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 value 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 position variable value. In CLOCK pulse units (see comments). As a signed 32-bit value. Allowed value range: [-2³¹...+(2³¹-1)].</p> <p>Tol Tolerance for the reference movement (see comments). As an unsigned 32-bit value. Allowed value range: [0...+(2³²-1)]. If Dir < 0 and/or Period = 0, then the value Tol = 0 is irrelevant, otherwise Tol = 0 causes stepper_init not to be executed (get_last_error return code RTC6_PARAM_ERROR).</p> <p>Enable ENABLE signal. As an unsigned 32-bit value. Allowed value range: [0...+(2³²-1)]. = 0: The ENABLE signal is reset. > 0: The ENABLE signal is set.</p> <p>WaitTime Defines when stepper_init, at the latest, returns (see comments). 1 bit equals 1 s. As an unsigned 32-bit value. Allowed value range: [0...+(2³²-1)].</p>

Ctrl Command	stepper_init
Comment	<ul style="list-style-type: none"> For programming the stepper motor signals, see Chapter 9.1.5 "Controlling Stepper Motors", page 283. stepper_init immediately stops all previously started movements of the stepper motor specified by the <code>No</code> parameter. The <code>ENABLE</code> signal <code>Enable</code> 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 <code>Clock</code> 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_stepper_status). If a limit switch gets activated during a movement, then the reference movement stops there. Afterward, the limit switch position is crossed 4× 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. The internal position variable (for the current position) gets 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. With <code>Pos = Tol</code>, the middle limit switch position corresponds to position 0. If no limit switch is found (for example, because no limit switch exists in the defined direction), then the stepper motor performs an infinite movement. stepper_init 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>Period1/Period2 = 0</code>). For more on this, see the Section "Terminating Infinite Movements", page 285. If <code>Period = 0</code>, <code>Dir < 0</code> and/or <code>WaitTime = 0</code>, then stepper_init returns straight away to immediately restore system control to the user program. You can then call get_stepper_status to check whether the reference movement has completed. During the seeking movement the status "Init" (see page 424) is set and during the terminating set-position movement to (limit switch + <code>Tol</code>) the status "Busy" (see page 424) is set. <code>Period = 0</code> and/or <code>Dir < 0</code> can be used as an emergency stop. Then a previously started stepper motor movement gets aborted, but no reference movement is performed. Here, too, the position variable gets set to the value <code>Pos</code>. The <code>DIRECTION</code> signal remains unchanged. If <code>Period = 0</code>, then status "Init" (see page 424) and status "Busy" (see page 424) get reset. No further clock pulses are outputted until <code>Period</code> is again set to a positive value.



Ctrl Command	stepper_init
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	–

Ctrl Command	stepper_rel
Function	Triggers set-position movements to the specified relative positions by both stepper motor output ports.
Call	<code>stepper_rel(dPos1, dPos2, WaitTime)</code>
Parameters	<p><code>dPos1</code> Relative set positions in CLOCK pulse units for stepper motor output port 1. As a signed 32-bit value. Allowed value range: $[-2^{31} \dots + (2^{31}-1)]$.</p> <p><code>dPos2</code> Relative set positions in CLOCK pulse units for stepper motor output port 2. As a signed 32-bit value. Allowed value range: $[-2^{31} \dots + (2^{31}-1)]$.</p> <p><code>WaitTime</code> Like stepper_abs.</p>
Comments	<ul style="list-style-type: none"> The set positions should be specified relative to the current position values. Otherwise, stepper_rel is identical to stepper_abs (see comments there).
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	stepper_abs , stepper_rel_list

Undelayed Short List Command	stepper_rel_list
Function	Like stepper_rel , but a list command and without <code>WaitTime</code> parameter.
Call	<code>stepper_rel_list(dPos1, dPos2)</code>
Parameters	<p><code>dPos1</code> Like stepper_rel.</p> <p><code>dPos2</code> Like stepper_rel.</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→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	stepper_rel

Ctrl Command	stepper_rel_no
Function	Triggers a set-position movement to the specified relative position at one stepper motor output port.
Call	<code>stepper_rel_no(No, dPos, WaitTime)</code>
Parameters	<p>No Like stepper_abs_no.</p> <p>dPos Relative position. In CLOCK pulse units. As a signed 32-bit value. Allowed value range: $[-2^{31} \dots + (2^{31}-1)]$.</p> <p>WaitTime Like stepper_abs_no.</p>
Comments	<ul style="list-style-type: none"> The set positions should be specified relative to the current position values (the - position value is correspondingly get newly set) and a set-position movement is only performed at the stepper motor output specified by <code>No</code>. Otherwise, <code>stepper_rel_no</code> is identical to stepper_abs (see comments there).
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	stepper_abs_no , stepper_abs , stepper_rel_no_list

Undelayed Short List Command	stepper_rel_no_list
Function	Like 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 port. As an unsigned 32-bit value. Allowed values: = 1: Stepper motor output port 1. = 2: Stepper motor output port 2. If the value is invalid, then <code>stepper_rel_no_list</code> is, already during loading, replaced by a list_nop (get_last_error return code <code>RTC6_PARAM_ERROR</code>).</p> <p>dPos Like 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 <code>stepper_rel_no_list</code> is delayed until the reference movement completes.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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>
Parameters	<p>No Number of the stepper motor output. As an unsigned 32-bit value. Allowed values: = 1: Stepper motor output port 1. = 2: Stepper motor output port 2. = 0, 3: Both stepper motor output ports. Only the two least-significant bits are evaluated.</p>
Comments	<ul style="list-style-type: none"> For programming the stepper motor signals, see Chapter 9.1.5 "Controlling Stepper Motors", page 283. If no stepper motor movement had been previously started at the specified stepper motor output port, then stepper_wait still needs 10 μs to execute, even though it otherwise has no effect. stepper_wait does not influence: <ul style="list-style-type: none"> the Signals for "Laser Active" Operation the List Status the List Execution Status
RTC4 → RTC6	New command.
RTC5 → RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	–

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"> • stop_execution deactivates the Signals for “Laser Active” Operation even if no list is active (here stop_execution has no other effects; here too: get_last_error return code RTC6_BUSY). • With stop_execution, the galvanometer scanners 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 galvanometer scanners should be set to a defined position using goto_xy. • The external start input ports are disabled, see Section “External Start”, page 289. • The Processing-on-the-fly correction is switched off. • The BUSY list status values (see read_status) and the BUSY list execution status-List Execution Status value (see get_status) are reset. • A list that has been interrupted by stop_execution cannot be resumed. It must instead be newly started (for example, by execute_list_pos). To only temporarily halt a list and later resume it, you can use pause_list. • stop_execution only affects the addressed RTC6 board. In a master/slave chain, stop_execution is not passed on to the slave boards. If all RTC6 boards of a master/slave chain are to be synchronously stopped, then simulate_ext_stop or an external stop signal must be called to any card of the master/slave chain, see also Chapter 6.6.3 “Master/Slave Operation”, page 123.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality. However: Master/slave change.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	get_startstop_info

Ctrl Command	stop_list
Function	Pauses execution of the list and deactivates the Signals for “Laser Active” Operation .
Call	<code>stop_list()</code>
Comments	<ul style="list-style-type: none"> • stop_list is synonymous with pause_list (see comments there).
RTC4→RTC6	Basically unchanged functionality. However: Additional PAUSED list execution status which is set by stop_list .
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	pause_list



Ctrl Command	stop_trigger
Function	Resets (to <code>Busy</code> = 0) the measurement session status that can be queried by <code>measurement_status</code> .
Call	<code>stop_trigger()</code>
Comments	<ul style="list-style-type: none"> stop_trigger is only needed if a measurement session has been started by <code>set_trigger</code> or <code>set_trigger4</code>, but not subsequently terminated by <code>set_trigger(Period = 0)</code> or <code>set_trigger4(Period = 0)</code>. Here, you can reset the measurement session status even when no list is active (see comments at <code>set_trigger</code>). stop_trigger is not executed (<code>get_last_error</code> return code <code>RTC6_BUSY</code>), if: <ul style="list-style-type: none"> the <code>BUSY</code> list execution status is set the <code>INTERNAL-BUSY</code> list execution status is set stop_trigger is even executed, if: <ul style="list-style-type: none"> a list has been paused by <code>set_wait</code> (<code>PAUSED</code> list execution status set)
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	<code>measurement_status</code> , <code>set_trigger</code> , <code>set_trigger4</code>

Ctrl Command	store_program													
Function	Standalone Functionality: Saves (deletes) data for automatic booting to (from) the NAND memory .													
Prerequisite	RTC6 Software Package ≥ V1.7.0 and BIOS-ETH ≥ 26.													
Call	Error = store_program(Mode)													
Parameters	<p>Mode = 0: Saves data for "Standalone Basic State" (see below).</p> <p>= 1: Erases the NAND memory content.</p> <p>> 1: Like = 0, and in addition some files for "Standalone Full State" (see below).</p> <p>As an unsigned 32-bit value.</p>													
Result	<p>Error code.</p> <p>As an unsigned 32-bit value.</p> <table> <tr> <td>Value</td> <td>Description</td> </tr> <tr> <td>0</td> <td>No error.</td> </tr> <tr> <td>1</td> <td>NAND memory not addressable.</td> </tr> <tr> <td>2</td> <td>NAND memory end reached early.</td> </tr> <tr> <td>3</td> <td>Data have not been or only partially stored in NAND memory.</td> </tr> <tr> <td>4</td> <td>Not an RTC6 Ethernet Board.</td> </tr> </table>		Value	Description	0	No error.	1	NAND memory not addressable.	2	NAND memory end reached early.	3	Data have not been or only partially stored in NAND memory .	4	Not an RTC6 Ethernet Board.
Value	Description													
0	No error.													
1	NAND memory not addressable.													
2	NAND memory end reached early.													
3	Data have not been or only partially stored in NAND memory .													
4	Not an RTC6 Ethernet Board.													
Comments	<ul style="list-style-type: none"> • store_program is only allowed with RTC6 Ethernet Boards. Otherwise, a get_last_error return code RTC6_TYPE_REJECTED is generated. • store_program is not executed (get_last_error return code RTC6_BUSY), if: <ul style="list-style-type: none"> – the BUSY list execution status is set – a list has been paused by set_wait (PAUSED list execution status set) • During the execution of store_program the 10 µs clock cycle of the DSP is interrupted: <ul style="list-style-type: none"> – Mode = 0 approx. 10 s – Mode > 1 approx. 60 s • Data for "Standalone Basic State" are: <ul style="list-style-type: none"> – RTC6ETH.out, RTC6RBF.rbf and RTC6DAT.dat • Data for "Standalone Full State" are: <ul style="list-style-type: none"> – Data for "Standalone Basic State" (see bullet above) – The required control commands, list commands and correction files • In case of an error, a get_last_error return code RTC6_FLASH_ERROR is generated. • See Chapter 16.7 "Standalone Functionality", page 890. 													
RTC4→RTC6	New command.													
RTC5→RTC6	New command.													
Version info	Available as of DLL 618, OUT 618, RBF 623.													
References	read_image_eth, write_image_eth													



Ctrl Command	store_timestamp_counter
Function	Saves the current 32-bit "Timestamp Counter" value.
Call	<code>store_timestamp_counter()</code>
Parameters	None.
Comments	<ul style="list-style-type: none"> See Chapter 8.12 "Time Measurements", page 280. The current 32-bit "Timestamp Counter" value is stored as time reference TimeStampStorage for wait_for_timestamp_counter.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 617, OUT 617, RBF 623.
References	store_timestamp_counter_list , wait_for_timestamp_counter

Undelayed Short List Command	store_timestamp_counter_list
Function	Like store_timestamp_counter , but a list command.
Call	<code>store_timestamp_counter_list()</code>
Parameters	None.
Comments	<ul style="list-style-type: none"> See store_timestamp_counter.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 617, OUT 617, RBF 623.
References	store_timestamp_counter , wait_for_timestamp_counter

Undelayed Short List Command	sub_call
Function	Causes an unconditional jump to an indexed subroutine.
Call	<code>sub_call(Index)</code>
Parameters	Index Index of the called indexed subroutine. As an unsigned 32-bit value. Allowed value range: [0...1023].
Comments	<ul style="list-style-type: none"> • sub_call reads the indexed subroutine's starting address from the internal management table based on the supplied index and then calls list_call (see also the comments there). list_call then triggers the jump to the subroutine. • sub_call 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. If applicable, a list_continue is executed. <p><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” (= $2^{32}-1$).</p> <ul style="list-style-type: none"> • 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>RTC6_PARAM_ERROR</code>). • Absolute Vector commands 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[*] Commands, arc commands and Jump commands or sub_call_abs must be used instead.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 commands and "Arc" commands receive an offset (corresponding to the current coordinates at the time of the call).
Call	sub_call_abs(Index)
Parameters	Index Index of the called indexed subroutine. As an unsigned 32-bit value. Allowed value range: [0...1023].
Comments	<ul style="list-style-type: none"> • sub_call_abs reads the indexed subroutine's starting address from the internal management table based on the supplied index and then calls list_call_abs (see also the comments there). list_call_abs 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→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	sub_call , sub_call_abs_cond

Undelayed Short List Command	sub_call_abs_cond						
Function	<i>Conditional call (AbsCall) of an indexed subroutine:</i> sub_call_abs_cond executes 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: $((\text{IOvalue AND Mask1}) = \text{Mask1}) \text{ AND } (((\text{not IOvalue}) \text{ AND Mask0}) = \text{Mask0})$ (= 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_abs_cond(Mask1, Mask0, Index)						
Parameters	<table> <tr> <td>Mask1</td> <td>16-bit mask. As an unsigned 32-bit value. Only the lower 16 bits are evaluated.</td> </tr> <tr> <td>Mask0</td> <td>See Mask1.</td> </tr> <tr> <td>Index</td> <td>Index of the called indexed subroutine. As an unsigned 32-bit value. Allowed value range: [0...1023].</td> </tr> </table>	Mask1	16-bit mask. As an unsigned 32-bit value. Only the lower 16 bits are evaluated.	Mask0	See Mask1.	Index	Index of the called indexed subroutine. As an unsigned 32-bit value. Allowed value range: [0...1023].
Mask1	16-bit mask. As an unsigned 32-bit value. Only the lower 16 bits are evaluated.						
Mask0	See Mask1.						
Index	Index of the called indexed subroutine. As an unsigned 32-bit value. Allowed value range: [0...1023].						
Comments	<ul style="list-style-type: none"> • See sub_call_abs. • See also Chapter 9.3.2 "Conditional Command Execution", page 294. 						
RTC4→RTC6	New command.						
RTC5→RTC6	Unchanged functionality.						
Version info	Available as of DLL 600, OUT 600, RBF 600.						
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	<code>sub_call_abs_repeat(Index, Number)</code>
Parameters	<p>Index Index of the to be called indexed subroutine (as with sub_call_abs).</p> <p>Number Number of repetitions. 0 is treated as 1. As an unsigned 32-bit value.</p>
Comments	<ul style="list-style-type: none"> • <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→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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> <code>sub_call_cond</code> executes <code>sub_call(Index)</code> , if the current IValue at the 16-bit digital input port of the EXTENSION 1 socket connector meets the following condition: $((IValue \text{ AND } Mask1) = Mask1) \text{ AND } (((\text{not } IValue) \text{ AND } Mask0) = Mask0)$ (= 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	<code>sub_call_cond(Mask1, Mask0, Index)</code>
Parameters	<p>Mask1 16-bit mask. As an unsigned 32-bit value. Only the lower 16 bits are evaluated.</p> <p>Mask0 See Mask1.</p> <p>Index Index of the to-be-called indexed subroutine. As an unsigned 32-bit value. Allowed value range: [0...1023].</p>
Comments	<ul style="list-style-type: none"> • See sub_call. • See also Chapter 9.3.2 "Conditional Command Execution", page 294.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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	sub_call_repeat(Index, Number)
Parameters	Index Index of the to be called indexed subroutine (as with sub_call). Number Number of repetitions. As an unsigned 32-bit value. Number = 0 is treated like Number = 1.
Comments	<ul style="list-style-type: none"> • sub_call(Index) is synonymous with <code>sub_call_repeat(Index, 1)</code>. • sub_call_repeat avoids an empty cycle at the repetition, which otherwise inevitably occurs with sub_call...sub_call or list_repeat...sub_call...list_until constructions. • By sub_call_repeat, for example, trajectories (see Glossary entry on page 29) from micro vector commands for runup curves and coast down curves can be seamlessly joined together with shapes from subroutines.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 (<i>Pos</i>) whose jump distance <i>Pos</i> (>1) is determined at runtime by the current value (<i>IOvalue</i>) 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 of the 16-bit digital input port to be evaluated for determining the jump distance. As an unsigned 32-bit value. Allowed value 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 value range: [0...15].
Comments	<ul style="list-style-type: none"> With invalid values of <i>MaskBits</i> or <i>ShiftBits</i> and with $(\text{MaskBits} + \text{ShiftBits}) > 16$, switch_ioport is replaced by a list_nop (get_last_error return code RTC6_PARAM_ERROR). The following applies: <i>Mask</i> = $((1 << \text{MaskBits}) - 1) << \text{ShiftBits}$ and <i>SwitchNo</i> = $(\text{Mask} \& \text{IOvalue}) >> \text{ShiftBits}$. Here, a list_jump_rel(<i>Pos</i>) with <i>Pos</i> = (<i>SwitchNo</i> + 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 (<i>Pos</i> = 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 Port and 16-Bit Digital Output Port", page 77 and Chapter 9.3.2 "Conditional Command Execution", page 294.
Example (Pascal)	<ul style="list-style-type: none"> It is assumed that the current value at the 16-bit digital input port is \$F152 at runtime: then switch_ioport(\$0008, \$0004) executes list_jump_rel(\$0016), that is, a relative list jump of length 22 list positions.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	list_jump_rel, list_jump_rel_cond

Ctrl Command	sync_slaves
Function	sync_slaves is no longer necessary as of DLL 614, OUT 614, RBF 619. Synchronizes all slave boards (connected in a master/slave chain with the slave connection of the addressed RTC6 board) stably to the 10 μ s clock cycle of the addressed RTC6 board.
Call	<code>sync_slaves()</code>
Comments	<ul style="list-style-type: none"> For usage of sync_slaves, see Chapter 6.6.3 "Master/Slave Operation", page 123. With RTC6 Software Package \geq V1.5.2, the actions described below for RTC6 Software Packages <V1.5.0 are no longer executed. RTC6 Software Packages <V1.5.0 (< RBF 619): <ul style="list-style-type: none"> SCANLAB recommends executing synchronization immediately after all boards have been initialized by load_program_file and load_correction_file. Otherwise, all involved boards (that is, 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 sync_slaves, you should neither apply external stop signals to the boards nor trigger External Starts (this is not automatically prevented). During the course of sync_slaves, a simulate_ext_stop is passed to 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). Users themselves are responsible to ensure that any running processes are not disrupted by that. After execution of sync_slaves, 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). sync_slaves (relating to synchronization) only affects RTC6 boards connected in a master/slave chain to the Master connector of the addressed RTC6 board. It does not affect the addressed board itself or any boards connected to the Slave connector of the addressed board. Therefore, if all slave boards of a master/slave chain is to be synchronized with the master board, then sync_slaves must address the master board of the master/slave chain. sync_slaves has no effect, if no board is connected to the Master connector of the addressed board. Synchronization of downstream slave boards by sync_slaves occurs even when the boards' BUSY list execution status or INTERNAL-BUSY list execution 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 sync_slaves has no effect. During the course of sync_slaves, all get_startstop_info error bits are cleared on all boards (including upstream boards) allocated to the user program.



Ctrl Command	sync_slaves
Comments (cont'd)	<ul style="list-style-type: none"> • sync_slaves is not executed (get_last_error return code RTC6_BUSY), if: <ul style="list-style-type: none"> – the BUSY list execution status of the addressed board is currently set – the INTERNAL-BUSY list execution status of the addressed board is set • sync_slaves is even executed, if: <ul style="list-style-type: none"> – a list has been paused by set_wait (PAUSED list execution status is set)
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality. However: Master/Slave functionality has been changed.
Version info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 615: sync_slaves has no function anymore.
References	get_master_slave , get_sync_status , master_slave_config



Ctrl Command	time_control_eth
Function	Sets a parameter to fine-tune the accuracy of the real-time clock.
Call	<code>time_control_eth(PPM)</code>
Parameters	PPM Deviation. As a 64-bit IEEE floating point value. Allowed value range: $PPM / 4.34 = [-64\dots+63]$
Comments	<ul style="list-style-type: none"> The execution of time_control_eth can take several 100 μs, see time_update. time_control_eth is not executed with RTC6 PCIe Boards. PPM means the deviation in parts per million. A positive value slows down the clock.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 612, ETH 612, RBF 617.
References	time_update



Normal List Command	time_fix
Function	Stores the current time and date of the RTC clock/calender in a cache for use with mark_date and mark_time .
Call	<code>time_fix()</code>
Comments	<ul style="list-style-type: none"> • time_fix is synonymous with time_fix_f_off with <code>FirstDay = 0</code> and <code>Offset = 0</code> (see comments there).
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 cache for use with mark_date and mark_time .
Call	<code>time_fix_f(FirstDay)</code>
Parameters	<code>FirstDay</code> Like time_fix_f_off .
Comments	<ul style="list-style-type: none"> • time_fix_f is synonymous with time_fix_f_off with <code>Offset = 0</code> (see comments there).
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	time_fix_f, time_fix_f_off, time_update, mark_time, mark_date

Normal List Command	time_fix_f_off
Function	Stores 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 in a cache.
Call	time_fix_f_off(FirstDay, Offset)
Parameters	FirstDay Defines the starting number for determining the Julian calendar day from the current date of the RTC calendar: Counting proceeds from FirstDay to FirstDay + 364 (+1 for leap years). As an unsigned 32-bit value.
	Offset Forward dating. In seconds. As an unsigned 32-bit value. Allowed value 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 RTC6 and PC time should be performed (for RTC6 boards, 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. 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 Offset = 0, then the current date and current time are fixed. One practical use of forward dating (Offset > 0) is for setting a date of expiry based on the current date. Backdating (Offset < 0) is not possible.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	time_fix , time_fix_f , time_update , mark_time , mark_date

Ctrl Command	time_update
Function	Sets the 24-hour clock and calendar of the RTC6 board to the current PC time.
Call	<code>time_update()</code>
Comments	<ul style="list-style-type: none"> • time_update must be called after each load_program_file, if 24-hour clock and calendar of the board are to be synchronized with the PC time. • 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. • The RTC6 Ethernet Board is a real-time clock. Therefore, time_update must be called only once. time_update takes several hundred μs. During this time the 10 μs clock cycle of the RTC6 Ethernet Board is interrupted. Therefore, time_update should not be called during list processing. The clock continues to run even if the power supply is switched off (> about 1 week). See also time_control_eth.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 612, ETH 612, RBF 617: real-time clock of the RTC6 Ethernet Board.
References	time_fix , time_fix_f , time_fix_f_off , mark_date , mark_time , time_control_eth

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 2D Image field .
Call	<code>timed_arc_abs(X, Y, Angle, T)</code>
Parameters	X Like arc_abs .
	Y Like arc_abs .
	Angle Like arc_abs .
	T Duration of the complete arc marking process. In μs . As a 64-bit IEEE floating point value. Allowed value range: [0...167,772,160]. 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 .
Comments	<ul style="list-style-type: none"> Unlike arc_abs, timed_arc_abs does not execute the marking process with the specified (by set_mark_speed or set_mark_speed_ctrl) mark speed. Instead, the speed (that is, the number of Microsteps) is adjusted so that the arc lasts as long as specified (see Chapter 8.9 "Timed Commands", page 273). The total marking time is (for $T \geq 5$) the sum of the specified (rounded) time and the set delays. See also comments on arc_abs.
RTC4→RTC6	New command. See arc_abs .
RTC5→RTC6	Unchanged functionality. See arc_abs .
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 2D Image field .
Call	timed_arc_rel(dx, dy, Angle, T)
Parameters	dx Like arc_rel .
	dy Like arc_rel .
	Angle Like arc_rel .
	T Duration of the complete arc marking process. In μ s. As a 64-bit IEEE floating point value. Allowed value range: [0...167,772,160]. 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 .
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, timed_arc_rel is analogous to timed_arc_abs (see the comments there).
RTC4→RTC6	<p>New command.</p> <p>In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for dx and dy by 16. The allowed value ranges decrease accordingly.</p>
RTC5→RTC6	Unchanged functionality. In addition: increased value range.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 2D Image field .						
Call	<code>timed_jump_abs(X, Y, T)</code>						
Parameters	<table> <tr> <td>X</td><td>Like jump_abs.</td></tr> <tr> <td>Y</td><td>Like jump_abs.</td></tr> <tr> <td>T</td><td>Duration of the complete jump vector. In μs. As a 64-bit IEEE floating point value. Allowed value range: [0...167,772,160]. 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.</td></tr> </table>	X	Like jump_abs .	Y	Like jump_abs .	T	Duration of the complete jump vector. In μs . As a 64-bit IEEE floating point value. Allowed value range: [0...167,772,160]. 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 .
X	Like jump_abs .						
Y	Like jump_abs .						
T	Duration of the complete jump vector. In μs . As a 64-bit IEEE floating point value. Allowed value range: [0...167,772,160]. 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 .						
Comments	<ul style="list-style-type: none"> Unlike jump_abs, timed_jump_abs does not execute the jump with the specified (by set_jump_speed or set_jump_speed_ctrl) jump speed. Instead, the speed (that is, the number of Microsteps) is adjusted so that the vector lasts as long as specified, see Chapter 8.9 "Timed Commands", page 273. The total jump time is (for $T \geq 5$) the sum of the specified (rounded) time and the set delays. See also comments on jump_abs. 						
RTC4→RTC6	Unchanged functionality. In addition: increased value range. In RTC4 Compatibility Mode , the RTC6 multiplies the specified values for X and Y by 16. The allowed value ranges decrease accordingly.						
RTC5→RTC6	Unchanged functionality. In addition: increased value range.						
Version info	Available as of DLL 600, OUT 600, RBF 600.						
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 the 3D image field .
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see select_cor_table), then timed_jump_abs_3d has the same effect as timed_jump_abs . However, split-up into Microsteps 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 Like jump_abs_3d.</p> <p>Y Like jump_abs_3d.</p> <p>Z Like jump_abs_3d.</p> <p>T Duration of the complete jump vector. In μs. As a 64-bit IEEE floating point value. Allowed value range: [0...167,772,160]. 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, timed_jump_abs_3d functions similarly to the timed_jump_abs command (see the comments there). See also comments on jump_abs_3d.
RTC4→RTC6	New command. See jump_abs_3d .
RTC5→RTC6	Unchanged functionality. In addition: increased value range.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 2D Image field .
Call	<code>timed_jump_rel(dX, dY, T)</code>
Parameters	<p><code>dX</code> Like jump_rel.</p> <p><code>dY</code> Like jump_rel.</p> <p><code>T</code> Duration of the complete jump vector. In μs. As a 64-bit IEEE floating point value. Allowed value range: [0...167,772,160]. 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.</p>
Comments	<ul style="list-style-type: none"> The coordinates for the jump vector end point are to be supplied as relative coordinates with respect to the current position. Otherwise, timed_jump_rel is identical to timed_jump_abs (see the comments there).
RTC4→RTC6	Unchanged functionality. In addition: increased value range. See jump_rel .
RTC5→RTC6	Unchanged functionality. In addition: increased value range.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 the 3D image field .
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see select_cor_table), then timed_jump_rel_3d has the same effect as timed_jump_rel . However, split-up into Microsteps is calculated like a 3D command and hence influences the effective jump speed in the xy plane.
Call	timed_jump_rel_3d(dx, dy, dz, T)
Parameters	dx Like jump_rel_3d . dy Like jump_rel_3d . dz Like jump_rel_3d . T Duration of the complete jump vector. In μs . As a 64-bit IEEE floating point value. Allowed value range: [0...167,772,160]. The parameter is rounded to an integer-multiple of 10. Out-of-range values are clipped. If $T < 5$, then timed_jump_rel_3d behaves like jump_rel_3d .
Comments	<ul style="list-style-type: none"> The coordinates for the jump vector end point are to be supplied as relative coordinates with respect to the current position. Otherwise, timed_jump_rel_3d is identical to timed_jump_abs_3d (see the comments there).
RTC4→RTC6	New command. See jump_rel_3d .
RTC5→RTC6	Unchanged functionality. In addition: increased value range.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 2D Image field .						
Call	<code>timed_mark_abs(X, Y, T)</code>						
Parameters	<table> <tr> <td>X</td><td>Like mark_abs.</td></tr> <tr> <td>Y</td><td>Like mark_abs.</td></tr> <tr> <td>T</td><td>Duration of the complete mark vector. In μs. As a 64-bit IEEE floating point value. Allowed value range: [0...167,772,160]. 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.</td></tr> </table>	X	Like mark_abs .	Y	Like mark_abs .	T	Duration of the complete mark vector. In μs . As a 64-bit IEEE floating point value. Allowed value range: [0...167,772,160]. 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 .
X	Like mark_abs .						
Y	Like mark_abs .						
T	Duration of the complete mark vector. In μs . As a 64-bit IEEE floating point value. Allowed value range: [0...167,772,160]. 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 .						
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) mark speed. Instead, the speed (that is, the number of Microsteps) is adjusted so that the vector lasts as long as specified (see Chapter 8.9 "Timed Commands", page 273). The total marking time is (for $T \geq 5$) the sum of the specified (rounded) time and the set delays. See also comments on mark_abs. 						
RTC4→RTC6	Unchanged functionality. In addition: increased value range. See mark_abs .						
RTC5→RTC6	Unchanged functionality. In addition: increased value range.						
Version info	Available as of DLL 600, OUT 600, RBF 600.						
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 the 3D image field .								
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see select_cor_table), then timed_mark_abs_3d has the same effect as timed_mark_abs . However, split-up into Microsteps is calculated like a 3D command and hence influences the effective mark speed in the xy plane.								
Call	timed_mark_abs_3d(X, Y, Z, T)								
Parameters	<table> <tr> <td>X</td><td>Like mark_abs_3d.</td></tr> <tr> <td>Y</td><td>Like mark_abs_3d.</td></tr> <tr> <td>Z</td><td>Like mark_abs_3d.</td></tr> <tr> <td>T</td><td>Duration of the complete mark vector. In μs. As a 64-bit IEEE floating point value. Allowed value range: [0...167,772,160]. 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.</td></tr> </table>	X	Like mark_abs_3d .	Y	Like mark_abs_3d .	Z	Like mark_abs_3d .	T	Duration of the complete mark vector. In μs . As a 64-bit IEEE floating point value. Allowed value range: [0...167,772,160]. 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 .
X	Like mark_abs_3d .								
Y	Like mark_abs_3d .								
Z	Like mark_abs_3d .								
T	Duration of the complete mark vector. In μs . As a 64-bit IEEE floating point value. Allowed value range: [0...167,772,160]. 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 .								
Comments	<ul style="list-style-type: none"> Except for the additional motion in the third dimension, timed_mark_abs_3d functions similarly to timed_mark_abs (see the comments there). See also comments on mark_abs_3d. 								
RTC4→RTC6	New command. See mark_abs_3d .								
RTC5→RTC6	Unchanged functionality. In addition: increased value range.								
Version info	Available as of DLL 600, OUT 600, RBF 600.								
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 2D Image field .
Call	<code>timed_mark_rel(dX, dY, T)</code>
Parameters	<p>dX Like mark_rel.</p> <p>dY Like mark_rel.</p> <p>T Duration of the complete mark vector. In μs. As a 64-bit IEEE floating point value. Allowed value range: [0...167,772,160]. 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).</p>
Comments	<ul style="list-style-type: none"> The coordinates for the mark vector end point are to be supplied as relative coordinates with respect to the current position. Otherwise, timed_mark_rel is analogous to timed_mark_abs (see the comments there).
RTC4→RTC6	Unchanged functionality. In addition: increased value range. See mark_rel .
RTC5→RTC6	Unchanged functionality. In addition: increased value range.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 the 3D image field .
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see select_cor_table), then timed_mark_rel_3d has the same effect as timed_mark_rel . However, split-up into Microsteps is calculated like a 3D command and hence influences the effective mark speed in the xy plane.
Call	timed_mark_rel_3d(dx, dy, dz, T)
Parameters	<p>dx Like mark_rel_3d.</p> <p>dy Like mark_rel_3d.</p> <p>dz Like mark_rel_3d.</p> <p>T Duration of the complete mark vector. In μs. As a 64-bit IEEE floating point value. Allowed value range: [0...167,772,160]. The parameter is rounded to an integer-multiple of 10. Out-of-range values are clipped. If $T < 5$, then timed_mark_rel_3d behaves like mark_rel_3d.</p>
Comments	<ul style="list-style-type: none"> The coordinates for the mark vector end point are to be supplied as relative coordinates with respect to the current position. Otherwise, timed_mark_rel_3d is identical to timed_mark_abs_3d (see the comments there).
RTC4→RTC6	New command.. See mark_rel_3d .
RTC5→RTC6	Unchanged functionality. In addition: increased value range.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 2D Image field and, simultaneously as well as linearly, changes the signal parameter selected by set_vector_ctrl to the specified value.								
Call	timed_para_jump_abs(X, Y, P, T)								
Parameters	<table> <tr> <td>X</td><td>Like para_jump_abs.</td></tr> <tr> <td>Y</td><td>Like para_jump_abs.</td></tr> <tr> <td>P</td><td>Like para_jump_abs.</td></tr> <tr> <td>T</td><td>Duration of the complete jump vector. In μs. As a 64-bit IEEE floating point value. Allowed value range: [0...167,772,160]. 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.</td></tr> </table>	X	Like para_jump_abs .	Y	Like para_jump_abs .	P	Like para_jump_abs .	T	Duration of the complete jump vector. In μs . As a 64-bit IEEE floating point value. Allowed value range: [0...167,772,160]. 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 .
X	Like para_jump_abs .								
Y	Like para_jump_abs .								
P	Like para_jump_abs .								
T	Duration of the complete jump vector. In μs . As a 64-bit IEEE floating point value. Allowed value range: [0...167,772,160]. 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 .								
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 (that is, the number of Microsteps) is adjusted so that the vector lasts as long as specified (see Chapter 8.9 "Timed Commands", page 273). 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 memory positions. 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 $10 \mu\text{s}$ clock, unless further (previous) short list commands induce a list_continue between the two parts. See also comments on para_jump_abs. 								
RTC4→RTC6	New command. See para_jump_abs .								
RTC5→RTC6	Unchanged functionality. In addition: increased value range.								
Version info	Available as of DLL 600, OUT 600, RBF 600.								
References	para_jump_abs , timed_jump_abs , jump_abs , timed_para_jump_rel , timed_para_jump_abs_3d								

Multiple List 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 the 3D image field and, simultaneously as well as linearly, changes the signal parameter selected by set_vector_control to the specified value.										
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see select_cor_table), then timed_para_jump_abs_3d has the same effect as timed_para_jump_abs . However, split-up into Microsteps is calculated like a 3D command and hence influences the effective jump speed in the xy plane.										
Call	<code>timed_para_jump_abs_3d(X, Y, Z, P, T)</code>										
Parameters	<table> <tr> <td>X</td> <td>Like para_jump_abs_3d.</td> </tr> <tr> <td>Y</td> <td>Like para_jump_abs_3d.</td> </tr> <tr> <td>Z</td> <td>Like para_jump_abs_3d.</td> </tr> <tr> <td>P</td> <td>Like para_jump_abs_3d.</td> </tr> <tr> <td>T</td> <td>Duration of the complete jump vector. In μs. As a 64-bit IEEE floating point value. Allowed value range: [0...167,772,160]. 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.</td> </tr> </table>	X	Like para_jump_abs_3d .	Y	Like para_jump_abs_3d .	Z	Like para_jump_abs_3d .	P	Like para_jump_abs_3d .	T	Duration of the complete jump vector. In μs . As a 64-bit IEEE floating point value. Allowed value range: [0...167,772,160]. 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 .
X	Like para_jump_abs_3d .										
Y	Like para_jump_abs_3d .										
Z	Like para_jump_abs_3d .										
P	Like para_jump_abs_3d .										
T	Duration of the complete jump vector. In μs . As a 64-bit IEEE floating point value. Allowed value range: [0...167,772,160]. 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 .										
Comments	<ul style="list-style-type: none"> Except for the additional motion in the third dimension, timed_para_jump_abs_3d functions similarly to timed_para_jump_abs (see the comments there). For $T \geq 5$, timed_para_jump_abs_3d occupies two list storage positions. The initial component executes as an undelayed short list command before the principal part (a normal list command). See also comments on para_jump_abs_3d. 										
RTC4→RTC6	New command. See para_jump_abs_3d .										
RTC5→RTC6	Unchanged functionality. In addition: increased value range.										
Version info	Available as of DLL 600, OUT 600, RBF 600.										
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 2D 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_rel(dx, dy, p, T)
Parameters	<p>dx Like para_jump_rel.</p> <p>dy Like para_jump_rel.</p> <p>p Like para_jump_rel.</p> <p>T Duration of the complete jump vector. In μs. As a 64-bit IEEE floating point value. Allowed value range: [0...167,772,160]. 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 end point are to be supplied as relative coordinates with respect to the current position. Otherwise, timed_para_jump_rel is analogous to timed_para_jump_abs (see the comments there).
RTC4→RTC6	New command. See para_jump_rel .
RTC5→RTC6	Unchanged functionality. In addition: increased value range.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	timed_para_jump_abs , para_jump_rel , timed_jump_rel , timed_para_jump_rel_3d

Multiple List 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 the 3D image field and, simultaneously as well as linearly, changes the signal parameter selected by set_vector_control to the specified value.
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see select_cor_table), then timed_para_jump_rel_3d has the same effect as timed_para_jump_rel . However, split-up into Microsteps is calculated like a 3D command and hence influences the effective jump speed in the xy plane.
Call	timed_para_jump_rel_3d(dX, dY, dZ, P, T)
Parameters	<p>dX Like para_jump_rel_3d.</p> <p>dY Like para_jump_rel_3d.</p> <p>dZ Like para_jump_rel_3d.</p> <p>P Like para_jump_rel_3d.</p> <p>T Duration of the complete jump vector. In μs. As a 64-bit IEEE floating point value. Allowed value range: [0...167,772,160]. 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 end point are to be supplied as relative coordinates with respect to the current position. Otherwise, timed_para_jump_rel_3d is analogous to timed_para_jump_abs_3d (see comments there). For $T \geq 5$, timed_para_jump_rel_3d occupies two list storage positions. The initial component executes as an undelayed short list command before the principal part (a normal list command).
RTC4→RTC6	New command. See para_jump_rel_3d .
RTC5→RTC6	Unchanged functionality. In addition: increased value range.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 2D 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	<table> <tr> <td>X</td><td>Like para_mark_abs.</td></tr> <tr> <td>Y</td><td>Like para_mark_abs.</td></tr> <tr> <td>P</td><td>Like para_mark_abs.</td></tr> <tr> <td>T</td><td>Duration of the complete mark vector. In μs. As a 64-bit IEEE floating point value. Allowed value range: [0...167,772,160]. 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.</td></tr> </table>	X	Like para_mark_abs .	Y	Like para_mark_abs .	P	Like para_mark_abs .	T	Duration of the complete mark vector. In μs . As a 64-bit IEEE floating point value. Allowed value range: [0...167,772,160]. 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 .
X	Like para_mark_abs .								
Y	Like para_mark_abs .								
P	Like para_mark_abs .								
T	Duration of the complete mark vector. In μs . As a 64-bit IEEE floating point value. Allowed value range: [0...167,772,160]. 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 .								
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) mark speed. Instead, the speed (that is, the number of Microsteps) is adjusted so that the vector lasts as long as specified (see Chapter 8.9 "Timed Commands", page 273). 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 for $T \geq 5$. 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 $10 \mu\text{s}$ clock, unless further (previous) short list commands induce a list_continue between the two parts. See also comments on para_mark_abs. 								
RTC4→RTC6	New command. See para_mark_abs .								
RTC5→RTC6	Unchanged functionality. In addition: increased value range.								
Version info	Available as of DLL 600, OUT 600, RBF 600.								
References	para_mark_abs , timed_mark_abs , mark_abs , timed_para_mark_rel , timed_para_mark_abs_3d								



Multiple List 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 the 3D image field and, simultaneously as well as linearly, changes the signal parameter selected by set_vector_control to the specified value.										
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see select_cor_table), then timed_para_mark_abs_3d has the same effect as timed_para_mark_abs . However, the split-up into Microsteps is calculated like a 3D command and hence influences the effective mark speed in the xy plane.										
Call	timed_para_mark_abs_3d(X, Y, Z, P, T)										
Parameters	<table> <tr> <td>X</td> <td>Like para_mark_abs_3d.</td> </tr> <tr> <td>Y</td> <td>Like para_mark_abs_3d.</td> </tr> <tr> <td>Z</td> <td>Like para_mark_abs_3d.</td> </tr> <tr> <td>P</td> <td>Like para_mark_abs_3d.</td> </tr> <tr> <td>T</td> <td>Duration of the complete mark vector. In μs. As a 64-bit IEEE floating point value. Allowed value range: [0...167,772,160]. 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.</td> </tr> </table>	X	Like para_mark_abs_3d .	Y	Like para_mark_abs_3d .	Z	Like para_mark_abs_3d .	P	Like para_mark_abs_3d .	T	Duration of the complete mark vector. In μs . As a 64-bit IEEE floating point value. Allowed value range: [0...167,772,160]. 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 .
X	Like para_mark_abs_3d .										
Y	Like para_mark_abs_3d .										
Z	Like para_mark_abs_3d .										
P	Like para_mark_abs_3d .										
T	Duration of the complete mark vector. In μs . As a 64-bit IEEE floating point value. Allowed value range: [0...167,772,160]. 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 .										
Comments	<ul style="list-style-type: none"> Except for the additional motion in the third dimension, timed_para_mark_abs_3d functions similarly to timed_para_mark_abs (see the comments there). timed_para_mark_abs_3d occupies two list storage positions for $T \geq 5$. The initial component executes as an undelayed short list command before the principal part (a normal list command). See also comments on para_mark_abs_3d. 										
RTC4→RTC6	New command. See para_mark_abs_3d .										
RTC5→RTC6	Unchanged functionality. In addition: increased value range.										
Version info	Available as of DLL 600, OUT 600, RBF 600.										
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 2D 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_rel(dx, dy, p, T)
Parameters	<p>dx Like para_mark_rel.</p> <p>dy Like para_mark_rel.</p> <p>p Like para_mark_rel.</p> <p>T Duration of the complete mark vector. In μs. As a 64-bit IEEE floating point value. Allowed value range: [0...167,772,160]. 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.</p>
Comments	<ul style="list-style-type: none"> The coordinates for the mark vector end point are to be supplied as relative coordinates with respect to the current position. Otherwise, timed_para_mark_rel is analogous to timed_para_mark_abs (see the comments there). See also comments on para_mark_rel.
RTC4→RTC6	New command. See para_mark_rel .
RTC5→RTC6	Unchanged functionality. In addition: increased value range.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	timed_para_mark_abs , para_mark_rel , timed_mark_rel , timed_para_mark_rel_3d

Multiple List 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 the 3D image field . Simultaneously varies the signal parameter selected by set_vector_control to the specified value.
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see select_cor_table), then timed_para_mark_rel_3d has the same effect as timed_para_mark_rel . However, split-up into Microsteps is calculated like a 3D command and hence influences the effective mark speed in the xy plane.
Call	timed_para_mark_rel_3d(dX, dY, dZ, P, T)
Parameters	<p>dX Like para_mark_rel_3d.</p> <p>dY Like para_mark_rel_3d.</p> <p>dZ Like para_mark_rel_3d.</p> <p>P Like para_mark_rel_3d.</p> <p>T Duration of the complete mark vector. In μs. As a 64-bit IEEE floating point value. Allowed value range: [0...167,772,160]. 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.</p>
Comments	<ul style="list-style-type: none"> The coordinates for the mark vector end point are to be supplied as relative coordinates with respect to the current position. Otherwise, timed_para_mark_rel_3d is analogous to timed_para_mark_abs_3d (see the comments there). timed_para_mark_rel_3d occupies two list storage positions for $T \geq 5$. The initial component executes as an undelayed short list command before the principal part (a normal list command). See also comments on para_mark_rel_3d.
RTC4→RTC6	New command. See para_mark_rel_3d .
RTC5→RTC6	Unchanged functionality. In addition: increased value range.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 Parameter: to-be-transformed position value. As a pointer to a signed 32-bit value.</p> <p>Returned parameter value: transformed position value. As a signed 32-bit value. The input values are overwritten.</p> <p>Sig2 Like Sig1 (analogously).</p>																												
Parameters	<p>Ptr Pointer (in C and C++ data type ULONG_PTR, an unsigned 32-bit value or unsigned 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 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. As an unsigned 32-bit value.</p> <p>The parameter's meaning is similar to that of get_transform (Sig1 corresponds to <code>Ptr1</code> and Sig2 to <code>Ptr2</code>).</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>Bit #6</td> <td>Reserved.</td> <td></td> </tr> <tr> <td>...</td> <td></td> <td></td> </tr> <tr> <td>Bit #31</td> <td></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.	Bit #6	Reserved.		...			Bit #31		
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.																											
Bit #6	Reserved.																												
...																													
Bit #31																													



Ctrl Command	transform												
Parameters (cont'd)	<p>Code (cont'd) If Bit #0 = 1, then one of the two supplied position values (specifiable as <code>Sig1</code> or <code>Sig2</code>) is backward transformed as the z coordinate:</p> <p>Bit #1 = 0: The value supplied by <code>Sig1</code> is backward transformed as the z coordinate (<code>Sig2</code> remains unchanged). Bit #1 = 1: The value supplied by <code>Sig2</code> is backward transformed as the z coordinate (<code>Sig1</code> remains unchanged).</p> <p>Bit #2 = 0: The offset to the focal length defined by <code>set_defocus</code> or <code>set_defocus_list</code> 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 <code>set_offset_xyz</code> or <code>set_offset_xyz_list</code> is backward transformed.</p> <p>Bit #5 Reserved.</p> <p>...</p> <p>Bit #31</p>												
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><code>Ptr</code> = <code>NULL</code> (no memory area specified).</td> </tr> <tr> <td>2</td> <td>No valid data at <code>Ptr</code> (<code>upload_transform</code> did not execute).</td> </tr> <tr> <td>3</td> <td>Erroneous data at <code>Ptr</code> (a corresponding error indication has been stored by <code>upload_transform</code>).</td> </tr> <tr> <td>4</td> <td>z axis inversion not possible.</td> </tr> </tbody> </table>	Value	Description	0	Success.	1	<code>Ptr</code> = <code>NULL</code> (no memory area specified).	2	No valid data at <code>Ptr</code> (<code>upload_transform</code> did not execute).	3	Erroneous data at <code>Ptr</code> (a corresponding error indication has been stored by <code>upload_transform</code>).	4	z axis inversion not possible.
Value	Description												
0	Success.												
1	<code>Ptr</code> = <code>NULL</code> (no memory area specified).												
2	No valid data at <code>Ptr</code> (<code>upload_transform</code> did not execute).												
3	Erroneous data at <code>Ptr</code> (a corresponding error indication has been stored by <code>upload_transform</code>).												
4	z axis inversion not possible.												
Comments	<ul style="list-style-type: none"> For backward transformation of position values see Chapter 8.1.3 "Monitoring the Positioning", page 213. The execution of <code>transform</code> must be preceded by a call to <code>upload_transform</code>. Additionally, position values should have been requested by <code>get_values</code>. If execution of <code>transform</code> results in an error (returned error code > 0), then no transformation occurs (<code>Sig1</code> and <code>Sig2</code> then remain unchanged). Errors also include <code>Ptr</code> = <code>NULL</code> (error code = 1) or errors resulting from prior, erroneous execution of <code>upload_transform</code> (error code = 3). If backward transformation of z values is requested (Code Bit #0 = 1), but only a 2D correction table has been 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 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 transform does not access any RTC6 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 RTC6 board on the PCIe bus). transform is not available as a multi-board command. The board-specific error variables LastError and AccError (see Chapter 6.8 "Error Handling", page 129) are neither generated nor altered by transform.
RTC4→RTC6	<p>New command.</p> <p>In the RTC4 Compatibility Mode, all back transformed values (including z values) are in the RTC6 20-bit range.</p>
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	upload_transform , get_transform , get_values



Ctrl Command	uart_config
Function	Configures the internal UART interface for the specified baud rate.
Call	<code>RealBaudRate = uart_config(BaudRate)</code>
Parameters	BaudRate Baud rate. As an unsigned 32-bit value. Allowed value range: [160 Bd...12.8 MBd].
Result	<code>RealBaudRate</code>
Comments	<ul style="list-style-type: none"> • uart_config extends rs232_config with a higher value range for the baud rate. • uart_config is synonymous with rs232_config, but returns the nearest possible actually used baud rate. • The default value is 9,600 baud. • The other RS-232 interface parameters cannot be altered (data bits: 8, start bits: 1, stop bits: 1, parity: none). • See also Chapter 4.6.5 "RS232 Socket Connector", page 82.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 611, OUT 611, RBF 616.
References	rs232_config , rs232_write_data , rs232_read_data



Ctrl Command	upload_transform
Function	Transfers from the RTC6 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: First scan head connector. = 2: Second scan head connector.</p> <p>Ptr Pointer (in C and C++ data type <code>ULONG_PTR</code>, an unsigned 32-bit value or unsigned 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> <p>Bit #0 =1: X gain = 0 (gain of automatic self-calibration for Galvanometer scanner 2).</p> <p>Bit #1 =1: Y gain = 0 (gain of automatic self-calibration for Galvanometer scanner 1).</p> <p>Bit #2 =1: The total matrix of the defined coordinate transformation is noninvertable.</p> <p>Bit #3 =1: No correction table assigned.</p> <p>Bit #4 =1: The ABC values (z axis) are noninvertable.</p> <p>Bit #5 =1: Error querying correction table.</p> <p>Bit #6 =1: Parameter error: invalid <code>HeadNo</code> or <code>Ptr</code> = 0.</p> <p>Bit #7 =1: Busy error, board has BUSY list execution status or INTERNAL-BUSY list execution status (<code>get_last_error</code> return code <code>RTC6_BUSY</code>).</p> <p>Bit #8 Reserved.</p> <p>...</p> <p>Bit #31</p>
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 Chapter 8.1.3 "Monitoring the Positioning", page 213). 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 528,520 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. 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 <code>RTC6_PARAM_ERROR</code>).



Ctrl Command	upload_transform
Comments (cont'd)	<ul style="list-style-type: none"> • upload_transform is not executed (get_last_error return code RTC6_BUSY), if: <ul style="list-style-type: none"> – the BUSY list execution status is set – the INTERNAL-BUSY list execution status is set • upload_transform is even executed, if: <ul style="list-style-type: none"> – a list has been paused by set_wait (PAUSED list execution status set) • During the runtime of upload_transform, External Starts are suppressed.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	transform , get_transform



Ctrl Command	verify_checksum
Function	Checks or creates the checksum of a correction file.
Call	<code>verify_checksum(Name)</code>
Parameters	<p>Name Name of the correction file. As a pointer to a \0-terminated ANSI string.</p>
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 has been 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 Loading of correction files, page 131 and set_verify). • The verify_checksum command is available even without explicit access rights to a particular RTC6 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 checksums for the output files. • The board-specific error variables <code>LastError</code> and <code>AccError</code> (see Chapter 6.8 "Error Handling", page 129) are neither generated nor altered by verify_checksum.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	set_verify

Normal List Command	<code>wait_for_1_axis</code>
Function	"Fly Extension" Command: Waits until the value for the specified Mode has been exceeded or underrun.
Call	<code>wait_for_1_axis(Value, Mode, WaitMode, LaserMode)</code>
Parameters	<p>Value Value to be waited for. As a signed 32-bit value.</p> <p>Mode Mode from Table 4, page 260. As an unsigned 32-bit value.</p> <p>WaitMode < 0: Undercutting. = 0: Equal. > 0: Overrun. The galvanometer scanner follow the object movement. WaitMode+16: The galvanometer scanner stop. As a signed 32-bit value.</p> <p>LaserMode = 0: The laser remains unchanged. > 0: The laser is switched off after a LaserOff Delay. As an unsigned 32-bit value.</p>
Comments	<ul style="list-style-type: none"> Being an "Fly Extension" Command, <code>wait_for_1_axis</code> must not be used mixed with "Classic" Processing-on-the-fly commands (see Footnote, page 241). See Chapter 8.6 "Processing-on-the-fly", page 241 and Section "Fly Extension" Commands, page 258. Depending on the set Mode, it can be waited for an encoder value or (any) McBSP value. In case of McBSP, a corresponding Processing-on-the-fly correction should be enabled. Make sure that the data is transferred in the correct format. With <code>wait_for_1_axis</code> and <code>wait_for_2_axes</code>, Mode 1...4 must not be specified. Instead, Mode 17...20 is to be used. Mode 17...18 is to be used with an scan system with SCANahead control, if waiting is yet to occur within PreviewTime. Outside of this, Mode 19...20 can be used. Mode 17...18 and Mode 19...20 are identical with intelliSCAN systems. LaserMode = 0: like before. LaserMode > 0: Laser is switched off after a LaserOff Delay. WaitMode is like Mode of wait_for_encoder_mode. WaitMode and WaitMode+16 differentiate the galvanometer scanner movement, not set_fly_2d and set_fly_x/set_fly_y. wait_for_encoder with automatic positions-dependent selection of direction is not supported. With an unallowed parameter value, <code>wait_for_1_axis</code> is replaced by a list_nop (get_last_error return code RTC6_PARAM_ERROR).



Normal List Command	wait_for_1_axis
Comments (cont'd)	<ul style="list-style-type: none"> The following command calls are executed in the same way: <ul style="list-style-type: none"> <code>wait_for_1_axis(Value, EncoderNo + 19, Mode, 0)</code> = <code>wait_for_encoder_mode(Value, EncoderNo, Mode)</code> and set_fly_2d session <code>wait_for_1_axis(Value, EncoderNo + 19, Mode+16, 0)</code> = <code>wait_for_encoder_mode(Value, EncoderNo, Mode)</code> and set_fly_x/set_fly_y session <code>wait_for_1_axis(Value, 6, Mode, 0)</code> = <code>wait_for_mcbsp(Axis, Value, Mode)</code> and set_fly_x_pos session <code>wait_for_1_axis(Value, Axis×4+10, Mode+16, 0)</code> = <code>wait_for_mcbsp(Axis, Value, Mode)</code> and set_fly_x_pos/set_fly_y_pos session
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 617, OUT 617, RBF 623.
References	wait_for_2_axes

Multiple List Command	wait_for_2_axes
Function	"Fly Extension" Command: Waits until the values for both modes (see Mode) are within or outside the specified range.
Call	<code>wait_for_2_axes(ModeX, MinValueX, MaxValueX, ModeY, MinValueY, MaxValueY, WaitMode, LaserMode)</code>
Parameters	<p>ModeX Mode from Table 4, page 260. As an unsigned 32-bit value.</p> <p>MinValueX Lower limit for the x axis Mode value to be waited for. As a signed 32-bit value.</p> <p>MaxValueX Upper limit for the x axis Mode value to be waited for. As a signed 32-bit value.</p> <p>ModeY Mode from Table 4, page 260. As an unsigned 32-bit value.</p> <p>MinValueY Lower limit for the y axis Mode value to be waited for. As a signed 32-bit value.</p> <p>MaxValueY Upper limit for the y axis Mode value to be waited for. As a signed 32-bit value.</p> <p>WaitMode ≥ 0: Within limit values. < 0: Outside limit values. The galvanometer scanner follow the object movement. WaitMode+16: The galvanometer scanners stop. As a signed 32-bit value.</p> <p>LaserMode = 0: The laser remains unchanged. > 0: The laser is switched off after a LaserOff Delay. As an unsigned 32-bit value.</p>
Comments	<ul style="list-style-type: none"> Being an "Fly Extension" Command, wait_for_2_axes must not be used mixed with "Classic" Processing-on-the-fly commands (see Footnote, page 241). See Chapter 8.6 "Processing-on-the-fly", page 241 and Section ""Fly Extension" Commands", page 258. wait_for_2_axes 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. Depending on the set Mode, it can be waited for an encoder value or (any) McBSP value to be inside or outside the limits. In case of McBSP, a corresponding Processing-on-the-fly correction should be enabled. Make sure that the data is transferred in the correct format. With wait_for_1_axis and wait_for_2_axes, Mode 1...4 must not be specified. Instead, Mode 17...20 is to be used. Mode 17...18 is to be used with an scan system with SCANahead control, if waiting is yet to occur within PreviewTime. Outside of this, Mode 19...20 can be used. Mode 17...18 and Mode 19...20 are identical with intelliSCAN systems.



Multiple List Command	wait_for_2_axes
Comments (cont'd)	<ul style="list-style-type: none"> • LaserMode = 0: like before. • LaserMode > 0: The laser is switched off after a LaserOff Delay. • WaitMode is like Mode of wait_for_encoder_mode. • WaitMode and WaitMode+16 differentiate the galvanometer scanner movement not set_fly_2d and set_fly_x/set_fly_y. • With an unallowed parameter value, wait_for_2_axes is replaced by a list_nop (get_last_error return code RTC6_PARAM_ERROR). • The following command calls are executed in the same way: <ul style="list-style-type: none"> – wait_for_2_axes(19, EncXmin, EncXmax, 20, EncYmin, EncYmax, 0, 0) = wait_for_encoder_in_range_mode(EncXmin, EncXmax, EncYmin, EncYmax, 0) and set_fly_2d session, intelliSCAN – wait_for_2_axes(19, EncXmin, EncXmax, 20, EncYmin, EncYmax, 16, 0) = wait_for_encoder_in_range_mode(EncXmin, EncXmax, EncYmin, EncYmax, 0) and set_fly_x/set_fly_y session, intelliSCAN – wait_for_2_axes(17, EncXmin, EncXmax, 18, EncYmin, EncYmax, 0, 0) = wait_for_encoder_in_range_mode(EncXmin, EncXmax, EncYmin, EncYmax, 2) and set_fly_2d session, excelliSCAN – wait_for_2_axes(17, EncXmin, EncXmax, 18, EncYmin, EncYmax, 16, 0) = wait_for_encoder_in_range_mode(EncXmin, EncXmax, EncYmin, EncYmax, 2) and set_fly_x/set_fly_y session, excelliSCAN
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 617, OUT 617, RBF 623.
References	wait_for_1_axis



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> <tr> <td>Value</td> <td>Count. As a signed 32-bit value. Allowed value 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 value 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 value 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"> • wait_for_encoder is synonymous with wait_for_encoder_mode with parameter Mode = 0 (see comments there). 				
RTC4→RTC6	New command.				
RTC5→RTC6	Unchanged functionality.				
Version info	Available as of DLL 600, OUT 600, RBF 600.				
References	wait_for_encoder_mode				

Multiple List 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 value. As a signed 32-bit value. Allowed value range: $[-2^{31} \dots + (2^{31}-1)]$.
	EncXmax Like EncXmin (analogously).
	EncYmin Like EncXmin (analogously).
	EncYmax Like EncXmin (analogously).
Comments	<ul style="list-style-type: none"> For usage of <code>wait_for_encoder_in_range</code>, see Chapter 9.3.3 "Synchronization by Encoder Signals", page 297 and Chapter 8.6.7 "Synchronizing Processing-on-the-fly Applications", page 251. <code>wait_for_encoder_in_range</code> 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 <code>EncXmin > EncXmax</code>, then both values are interchanged. If <code>EncYmin > EncYmax</code>, then both values are interchanged. If no encoder-based Processing-on-the-fly correction is active, then <code>wait_for_encoder_in_range</code> merely creates a waiting period (without galvanometer scanner motion). If <code>EncXmin = EncXmax</code> (or <code>EncYmin = EncYmax</code>), then waiting until a specific encoder value is possible. <code>wait_for_encoder_in_range</code> is available even if the Option Processing-on-the-fly is not enabled. <code>wait_for_encoder_in_range</code> 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 Signals for "Laser Active" Operation (for example, a <code>list_nop</code>). 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 Chapter 8.6.7 "Synchronizing Processing-on-the-fly Applications", page 251: They move with <code>set_fly_2d</code>, but otherwise remain stationary.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	wait_for_encoder_mode , park_position , park_return



Multiple List Command	wait_for_encoder_in_range_mode
Function	Waits until both encoder counters simultaneously lie within the specified range (including limits).
Call	<code>wait_for_encoder_in_range_mode(EncXmin, EncXmax, EncYmin, EncYmax, Mode)</code>
Parameters	<p>EncXmin Limit value. As a signed 32-bit value. Allowed value range: $[-2^{31} \dots + (2^{31}-1)]$.</p> <p>EncXmax Like EncXmin (analogously).</p> <p>EncYmin Like EncXmin (analogously).</p> <p>EncYmax Like EncXmin (analogously).</p> <p>Mode Mode. As a signed 32-bit value. = 0 or 1: Waits for direct encoder values ("classical behavior"). = 2: Waits for encoder values that are expected to be present in (set_scanahead_params parameter) PreviewTime (SCANAhead system behavior).</p>
Comments	<ul style="list-style-type: none"> For usage, see Chapter 8.6.7 "Synchronizing Processing-on-the-fly Applications", page 251 and Chapter 9.3.3 "Synchronization by Encoder Signals", page 297 . See also comments on wait_for_encoder_in_range. Use Mode like in wait_for_encoder_mode: <ul style="list-style-type: none"> – 0, 1 for intelliSCAN or SCANAhead systems outside an marking process – 2 for SCANAhead systems during an marking process wait_for_encoder_in_range is synonymous with <code>wait_for_encoder_in_range_mode(Mode = ,0)</code>.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 612, OUT 612, RBF 617.
References	wait_for_encoder_mode

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 value 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>Mode. As a 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 value 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	Mode. As a 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 value 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	Mode. As a 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 <code>wait_for_encoder_mode</code>, see Chapter 9.3.3 "Synchronization by Encoder Signals", page 297. If <code>Mode</code> = 0, ensure that the size and sign of the parameter <code>Value</code> is appropriate for the counting direction of the selected encoder (for external triggering, this corresponds to the workpiece's direction of motion). If <code>Value</code> > 0, <code>wait_for_encoder_mode</code> waits for overstepping, otherwise for understepping. If <code>Value</code> is positive and already less than the current encoder count, then <code>wait_for_encoder_mode</code> waits for a complete traversal of the counter (likewise if <code>Value</code> is negative and larger than the current encoder count). At a 1 MHz counter rate, this can take up to approx. 36 minutes! If <code>Mode</code> ≠ 0, then <code>wait_for_encoder_mode</code> waits for overstepping/understepping of the <code>Value</code> parameter independently of the current position and direction of motion. If <code>EncoderNo</code> > 1, then <code>wait_for_encoder_mode</code> is replaced with a list_nop (get_last_error return code <code>RTC6_PARAM_ERROR</code>). If no encoder-based Processing-on-the-fly correction is active, then <code>wait_for_encoder_mode</code> 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.). <code>wait_for_encoder_mode</code> is available even if the Option Processing-on-the-fly is not enabled. For <code>Mode</code> = 0, <code>wait_for_encoder_mode</code> is synonymous with wait_for_encoder. <code>wait_for_encoder_mode</code> does not alter the laser control signals. If you want the laser off during the wait, then <code>wait_for_encoder_mode</code> must be preceded by some other command that switches off the Signals for "Laser Active" Operation (for example, a list_nop). 						



Normal List Command	<code>wait_for_encoder_mode</code>
Comments (cont'd)	<ul style="list-style-type: none"> 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 Chapter 8.6.7 "Synchronizing Processing-on-the-fly Applications", page 251: They move with <code>set_fly_2d</code>, but otherwise remain stationary. Mode = 0 or ± 1 waits for direct encoder values: "classic" behavior. Mode = ± 2 waits for encoder values that are expected to be present in (set_scanahead_params parameter) <small>PreviewTime: SCANAhead system behavior.</small>
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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 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 is 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 Limit value. As a signed 32-bit value.</p> <p>Mode Mode. As a 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 usage of wait_for_mcbsp, see Chapter 9.3.4 "Synchronization and Online Positioning by McBSP Signals", page 299. wait_for_mcbsp is comparable to wait_for_encoder_mode, but the McBSP 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 Section "Correction via McBSP Interface", page 244. 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, wait_for_mcbsp is replaced by list_nop (get_last_error return code RTC6_PARAM_ERROR). If neither set_fly_x_pos, set_fly_y_pos nor set_fly_rot_pos are activated, then wait_for_mcbsp 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.). wait_for_mcbsp is available even if the Option Processing-on-the-fly is not enabled. wait_for_mcbsp does not alter the laser control signals. If you want the laser off during the wait, then wait_for_mcbsp must be preceded by some other command that switches off the Signals for "Laser Active" Operation (for example, a list_nop).
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	wait_for_encoder_mode



Normal List Command	wait_for_timestamp_counter
Function	Waits for a 32-bit "Timestamp Counter" value.
Call	<code>wait_for_timestamp_counter(TimeStampCounter)</code>
Parameters	TimeStampCounter 32-bit offset to wait for. As an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> • wait_for_timestamp_counter waits until the current 32-bit "Timestamp Counter" has reached the value <code>TimeStampStorage</code> (from store_timestamp_counter / store_timestamp_counter_list) + TimeStampCounter. • Delayed short list commands are executed first. • This allows absolute time references from the point in time <code>TimeStampStorage</code> to be established when loading a list. Accuracy: 10 μs. • If the specified time has already passed when wait_for_timestamp_counter is reached, a full 32-bit "Timestamp Counter" cycle is waited for (duration: approx. 12 hours). This can be cancelled by stop_execution or /STOP. • wait_for_timestamp_counter is synonymous with wait_for_timestamp_counter_mode(Mode > 2). • Synchronization with other RTC6 commands, such as wait_for_encoder or conditional commands, depends on external hardware. • See Chapter 8.12 "Time Measurements", page 280.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 617, OUT 617, RBF 623.
References	store_timestamp_counter , store_timestamp_counter_list , wait_for_timestamp_counter_mode

Normal List Command	wait_for_timestamp_counter_long
Function	Waits for an absolute 64-bit "Timestamp Counter" value.
Call	<code>wait_for_timestamp_counter_long(WaitCounterL, WaitCounterH, MaxWaitTime, Mode)</code>
Parameters	<p>WaitCounterL Lower part of the 64-bit "Timestamp Counter" value to wait for. As an unsigned 32-bit value.</p> <p>WaitCounterH Upper part of the 64-bit "Timestamp Counter" value to wait for. As an unsigned 32-bit value.</p> <p>MaxWaitTime Longest waiting time. As an unsigned 32-bit value.</p> <p>Mode Routines after a timeout. As an unsigned 32-bit value.</p> <ul style="list-style-type: none"> = 0: Ignore the RTC6 command. Continue with next list command. = 1: Abort list by stop_execution. = 2: Abort list by simulate_ext_stop including forward to all slaves, see Chapter 6.6.3 "Master/Slave Operation", page 123 and Chapter 9.3.1 "Starting and Stopping Lists by External Control Signals and Master/Slave Synchronization", page 288. > 2: Wait endlessly until an External Stop occurs.
Comments	<ul style="list-style-type: none"> • wait_for_timestamp_counter_long waits until the current 64-bit "Timestamp Counter" has reached the following value: <code>WaitCounterL + (WaitCounterH << 32)</code> • This allows absolute time references to a synchronization point in time defined in the user program to be established when loading a list. Accuracy: 10 μs. • An error flag is set, if: <ul style="list-style-type: none"> – The specified point in time has already passed when wait_for_timestamp_counter_long is reached – The requested waiting time is beyond the running 64-bit "Timestamp Counter" by more than <code>MaxWaitTime</code>. The error flag can be read out by get_startstop_info(Bit #5). • The <code>Mode</code> parameter determines how to proceed in case of a timeout. • Synchronization with other RTC6 commands, such as wait_for_encoder or conditional commands, depends on external hardware. • See Chapter 8.12 "Time Measurements", page 280.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 624, OUT 624.
References	get_timestamp_long

Normal List Command	wait_for_timestamp_counter_mode
Function	Like wait_for_timestamp_counter . In addition, it can be defined what should happen after a timeout.
Call	<code>wait_for_timestamp_counter_mode(TimeStampCounter, Mode)</code>
Parameters	<p>TimeStampCounter 32-Bit offset to wait for. As an unsigned 32-bit value.</p> <p>Mode Routines after a timeout. As an unsigned 32-bit value.</p> <ul style="list-style-type: none"> = 0: Ignore the RTC6 command. Continue with next list command. = 1: Abort list by stop_execution. = 2: Abort list by simulate_ext_stop including forward to all slaves, see Chapter 6.6.3 "Master/Slave Operation", page 123 and Chapter 9.3.1 "Starting and Stopping Lists by External Control Signals and Master/Slave Synchronization", page 288. > 2: Wait for a full 32-bit "Timestamp Counter" cycle (as with wait_for_timestamp_counter).
Comments	<ul style="list-style-type: none"> • wait_for_timestamp_counter waits until the current 32-bit "Timestamp Counter" has reached the value <code>TimeStampStorage</code> (from store_timestamp_counter / store_timestamp_counter_list) + <code>TimeStampCounter</code>. • Delayed short list commands are executed first. • This allows absolute time references with an accuracy of $10 \mu\text{s}$ from the point in time <code>TimeStampStorage</code> to be established when loading a list. Accuracy: $10 \mu\text{s}$. • An error flag is set, if: <ul style="list-style-type: none"> – The specified point in time has already passed when wait_for_timestamp_counter_mode is reached The error flag can be read out by get_startstop_info(Bit #5). The <code>Mode</code> parameter determines how to proceed in case of a timeout. • wait_for_timestamp_counter is synonymous with wait_for_timestamp_counter_mode(<code>Mode</code> > 2). • Synchronization with other RTC6 commands, such as wait_for_encoder or conditional commands, depends on external hardware. • See Chapter 8.12 "Time Measurements", page 280.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 619, OUT 619, RBF 624.
References	store_timestamp_counter , store_timestamp_counter_list , wait_for_timestamp_counter



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>
Parameters	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 282.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	fwrite_8bit_port_list

Delayed Short List Command	write_8bit_port_list
Function	Like write_8bit_port , but a list command.
Call	<code>write_8bit_port_list(Value)</code>
Parameters	Value Like write_8bit_port .
Comments	<ul style="list-style-type: none"> • See write_8bit_port. • As of version DLL 602, OUT 602: When the 8-Bit digital output (for example, for laser power) is to be executed synchronous to the laser switch times, use set_laser_power(2, Value).
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	fwrite_8bit_port , set_laser_power , write_port_list



Ctrl Command	write_abc_to_file								
Function	Like write_abc_to_file_20b . However, for a focus length value l in RTC4 compatibility range $[-32,768\dots+32,767]$.								
Call	ErrorNo = write_abc_to_file(Name, A, B, C)								
Parameters	<table> <tr> <td>Name</td> <td>Like write_abc_to_file_20b.</td> </tr> <tr> <td>A</td> <td>Coefficient A of the parabolic function $z_{out} = A + Bl + Cl^2$ which is used for calculating the Z output values (focus length value l in the RTC4 compatibility range $[-32,768\dots+32,767]$). As a 64-bit IEEE floating point value. Allowed value range:: see load_z_table.</td> </tr> <tr> <td>B</td> <td>Like A (analogously).</td> </tr> <tr> <td>C</td> <td>Like A (analogously).</td> </tr> </table>	Name	Like write_abc_to_file_20b .	A	Coefficient A of the parabolic function $z_{out} = A + Bl + Cl^2$ which is used for calculating the Z output values (focus length value l in the RTC4 compatibility range $[-32,768\dots+32,767]$). As a 64-bit IEEE floating point value. Allowed value range:: see load_z_table .	B	Like A (analogously).	C	Like A (analogously).
Name	Like write_abc_to_file_20b .								
A	Coefficient A of the parabolic function $z_{out} = A + Bl + Cl^2$ which is used for calculating the Z output values (focus length value l in the RTC4 compatibility range $[-32,768\dots+32,767]$). As a 64-bit IEEE floating point value. Allowed value range:: see load_z_table .								
B	Like A (analogously).								
C	Like A (analogously).								
Result	Like write_abc_to_file_20b .								
Comments	<ul style="list-style-type: none"> Like write_abc_to_file_20b. write_abc_to_file_20b(A × 16, B, C × 1/16) is synonymous with write_abc_to_file(A, B, C). 								
RTC4→RTC6	New command.								
RTC5→RTC6	Unchanged functionality.								
Version info	Available as of DLL 600, OUT 600, RBF 600.								
References	read_abc_from_file								

Ctrl Command	write_abc_to_file_20b	
Function	Writes the ABC values directly into a specified correction file on the PC. For a focus length value <i>l</i> in the RTC6 20-bit range [-524,288...+524,287].	
Call	ErrorNo = write_abc_to_file_20b(Name, A, B, C)	
Parameters	<p>Name Name of the correction file. As a pointer to a \0-terminated ANSI string.</p> <p>A Coefficient A of the parabolic function $z_{out} = A + B/l + C/l^2$ which is used for calculating the Z output values (focus length value <i>l</i> in the RTC6 20-bit range [-524,288...+524,287]). As a 64-bit IEEE floating point value. Allowed value range: see load_z_table_20b.</p> <p>B Like A (analogously).</p> <p>C Like A (analogously).</p>	
Result	<p>ErrorNo Error code. As an unsigned 32-bit value.</p> <p>0 No error.</p> <p>1 A exceeded the maximum allowed value.</p> <p>2 A undercut the minimum allowed value.</p> <p>4 B exceeded the maximum allowed value.</p> <p>8 B undercut the minimum allowed value.</p> <p>16 C exceeded the maximum allowed value.</p> <p>32 C undercut the minimum allowed value.</p> <p>3 File-open error (empty string, file not found etc.).</p> <p>12 File error (checksum could not be determined, file corrupt).</p>	
Comments	<ul style="list-style-type: none"> • <code>write_abc_to_file_20b</code> is available even without explicit access rights to a specific RTC6 board. • <code>write_abc_to_file_20b</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 129) are neither generated nor altered by <code>write_abc_to_file_20b</code>. • ABC values are not outputted with error code $\neq 0$ or $\neq 12$. • <code>write_abc_to_file_20b(A, B, C)</code> is synonymous with <code>write_abc_to_file(A × 1/16, B, C × 16)</code>. 	
RTC4→RTC6	New command.	
RTC5→RTC6	New command.	
Version info	Available as of DLL 631 , OUT 632 .	
References	read_abc_from_file_20b	



Ctrl Command	write_da_1
Function	See write_da_x .
Call	<code>write_da_1(Value)</code>
Parameters	<p>Value 12-bit output value for the ANALOG OUT1 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>
RTC4→RTC6	Unchanged functionality. In RTC4 Compatibility Mode : like write_da_x .
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	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>
Parameters	Value Like write_da_1 .
RTC4→RTC6	Unchanged functionality. In RTC4 Compatibility Mode : like write_da_x .
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	write_da_x_list



Ctrl Command	write_da_2
Function	See write_da_x .
Call	<code>write_da_2(Value)</code>
Parameters	<p>Value 12-bit output value for the ANALOG OUT2 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>
RTC4→RTC6	Unchanged functionality. In RTC4 Compatibility Mode : like write_da_x .
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	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>
Parameters	Value Like write_da_2 .
RTC4→RTC6	Unchanged functionality. In RTC4 Compatibility Mode : like write_da_x .
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	write_da_x_list

Ctrl Command	write_da_x
Function	Writes an output value to one of the analog output ports of the RTC6.
Call	<code>write_da_x(x, Value)</code>
Parameters	<p>x Number of the analog output port. As an unsigned 32-bit value. Allowed values: [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 Chapter 9.1.4 "12-Bit Analog Output Port 1 and 2", page 282. The output range of the analog output ports is 0 V...10 V. For $x < 1$ or $x > 2$, write_da_x is ignored (get_last_error return code RTC6_PARAM_ERROR). write_da_1 / write_da_2 can be used alternatively to write_da_x (without parameter x).
RTC4→RTC6	Unchanged functionality. In RTC4 Compatibility Mode , the RTC6 multiplies the specified value by 4.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	write_da_x_list



Delayed Short List Command	write_da_x_list
Function	Like write_da_x , but a list command.
Call	<code>write_da_x_list(x, Value)</code>
Parameters	<code>x</code> Like write_da_x .
	<code>Value</code> Like write_da_x .
Comments	<ul style="list-style-type: none"> For $x < 1$ or $x > 2$, <code>write_da_x_list</code> is replaced with a list_nop (get_last_error return code <code>RTC6_PARAM_ERROR</code>). As of version DLL 602, OUT 602: When the ANALOG OUT output port (for example, for the laser power) is to be executed synchronous to the laser switch times, use <code>set_laser_power(x-1, Value)</code>.
RTC4→RTC6	Unchanged functionality. In RTC4 Compatibility Mode : like write_da_x .
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	write_da_x , set_laser_power , write_port_list

Ctrl Command	write_hi_pos
Function	Writes the specified values to the Flash memory of the RTC6 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: First scan head connector. = 2: Second scan head connector.</p> <p>X1 X1 reference position. In bits. As a signed 32-bit value.</p> <p>X2 Like X1 (analogously).</p> <p>Y1 Like X1 (analogously).</p> <p>Y2 Like X1 (analogously).</p>
Result	<p>Error code. As an unsigned 32-bit value.</p> <p>0 Kein Fehler.</p> <p>Bit #0 =1: Wrong HeadNo.</p> <p>Bit #1 =1: Wrong sensor position for X1.</p> <p>Bit #2 =1: Wrong sensor position for X2.</p> <p>Bit #3 =1: Wrong sensor position for Y1.</p> <p>Bit #4 =1: Wrong sensor position for Y2.</p> <p>Bit #5 =1: Invalid ASC version.</p> <p>Bit #6 =1: Download failed. The values have possibly not been saved.</p>
Comments	<ul style="list-style-type: none"> • write_hi_pos is used in cases when a scan head is moved from RTC6 board "A" to RTC6 board "B" in order to transfer its Home-In reference positions from RTC6 board "A" to RTC6 board "B". • Use get_hi_pos(HeadNo) to read out the Home-In reference positions of RTC6 board "A" (only after init_rtc6_dll, see the get_hi_pos command description). • 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 RTC6 board "B". A cross-check with get_auto_cal(HeadNo) needs to return 100. Otherwise, write_hi_pos would return the error Bit #5 = 1. If required, connect the scan head to RTC6 board "B" and execute auto_cal(HeadNo, 4) before executing write_hi_pos. • write_hi_pos takes several hundred μs. During this time, the 10 μs clock cycle of RTC6 boards is interrupted.



Ctrl Command	write_hi_pos
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	get_hi_pos , get_auto_cal , auto_cal

Ctrl Command	write_image_eth		
Function	Standalone Functionality : Reads out data for automatic booting from a binary file on the PC and saves it to the NAND memory .		
Prerequisite	RTC6 Software Package ≥ V1.7.0 and BIOS-ETH ≥ 26.		
Call	<code>Result = write_image_eth(Name)</code>		
Parameters	<table> <tr> <td>Name</td> <td>Name of the binary file. As a pointer to a \0-terminated ANSI string.</td> </tr> </table>	Name	Name of the binary file. As a pointer to a \0-terminated ANSI string.
Name	Name of the binary file. As a pointer to a \0-terminated ANSI string.		
Result	<table> <tr> <td>Result</td> <td>Error code. Like Result of read_image_eth. As an unsigned 32-bit value.</td> </tr> </table>	Result	Error code. Like Result of read_image_eth . As an unsigned 32-bit value.
Result	Error code. Like Result of read_image_eth . As an unsigned 32-bit value.		
Comments	<ul style="list-style-type: none"> • write_image_eth is not executed (get_last_error return code RTC6_BUSY), if: <ul style="list-style-type: none"> – the BUSY list execution status is set – the INTERNAL-BUSY list execution status is set • If the Name cannot be opened, a get_last_error return code RTC6_PARAM_ERROR is generated. • During the execution of read_image_eth the 10 µs clock cycle of the DSP is interrupted for up to 2 minutes. • write_image_eth is only allowed with RTC6 Ethernet Boards. Otherwise, a get_last_error return code RTC6_TYPE_REJECTED is generated. • See Chapter 16.7 "Standalone Functionality", page 890. 		
RTC4→RTC6	New command.		
RTC5→RTC6	New command.		
Version info	Available as of DLL 618, OUT 618, RBF 623.		
References	read_image_eth , store_program		



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>
Parameters	<p>Value 16-bit output value (DIGITAL OUT0...DIGITAL OUT15). As an unsigned 32-bit value. Only the least significant 16 bits are evaluated.</p>
Comments	<ul style="list-style-type: none"> Use <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 digital input port. <code>write_io_port_mask</code> and <code>write_io_port_mask_list</code> also allow changing selectable bits of the 16-bit digital output port. See also Chapter 9.1.1 "16-Bit Digital Output Port", page 281.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
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	Like write_io_port , but a list command.
Call	<code>write_io_port_list(Value)</code>
Parameters	Value Like write_io_port .
Comments	<ul style="list-style-type: none"> See write_io_port. As of version DLL 602, OUT 602: When the 16-Bit digital output (for example, for laser power) is to be executed synchronous to the laser switch times, use <code>set_laser_power(3, Value).</code>
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version info	Available as of DLL 600, OUT 600, RBF 600.
References	write_io_port , set_laser_power , write_port_list

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	<code>Value</code>	16-bit output value (DIGITAL OUT0...DIGITAL OUT15). As an unsigned 32-bit value. Only the least significant 16 bits are evaluated.
	<code>Mask</code>	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, that is, 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 281. 	
RTC4→RTC6	New command.	
RTC5→RTC6	Unchanged functionality.	
Version info	Available as of DLL 600, OUT 600, RBF 600.	
References	write_io_port , write_io_port_mask_list	

Delayed Short List Command	write_io_port_mask_list	
Function	Like <code>write_io_port_mask</code> , but a list command.	
Call	<code>write_io_port_mask_list(Value, Mask)</code>	
Parameters	<code>Value</code>	Like <code>write_io_port_mask</code> .
	<code>Mask</code>	Like <code>write_io_port_mask</code> .
Comments	<ul style="list-style-type: none"> See <code>write_io_port_mask</code>. <code>write_io_port_mask_list</code> cannot be used for laser power control, see <code>write_io_port_list</code>. 	
RTC4→RTC6	New command.	
RTC5→RTC6	Unchanged functionality.	
Version info	Available as of DLL 600, OUT 600, RBF 600.	
References	write_io_port_mask , write_io_port_list , set_laser_power	

Undelayed Short List Command	write_port_list
Function	Writes a value to a digital or analog output port.
Call	<code>write_port_list(Port, Value, NoDelay)</code>
Parameters	<p>Port Output port. As an unsigned 32-bit value. Allowed values: = 0: ANALOG OUT1 output port. See also Section "12-Bit Analog Output Port 1 and 2", page 73. = 1: ANALOG OUT2 output port. See also Section "12-Bit Analog Output Port 1 and 2", page 73. = 2: 8-bit digital output port (EXTENSION 2 Socket Connector). See also Section "8-Bit Digital Output Port", page 80. = 3: 16-bit digital output port (EXTENSION 1 Socket Connector). See also Section "16-Bit Digital Input Port and 16-Bit Digital Output Port", page 77. = 4: 2-bit digital output port (LASER Connector). See also Section "2-Bit Digital Output Port", page 73. > 4: write_port_list is not executed <code>(get_last_error</code> return code <code>RTC6_PARAM_ERROR</code><code>).</code></p>
Value	<p>Output value. As an unsigned 32-bit value. Allowed values: For Port = 0: 12-bit values[0...4095]. For Port = 1: 12-bit values[0...4095]. For Port = 2: 8-bit values [0...255]. For Port = 3: 16-bit values [0...65,535]. For Port = 4: 2-bit values [0...3]. Out-of-range values are clipped to the boundary values. <code>get_last_error</code> return code <code>RTC6_PARAM_ERROR</code>.</p>
NoDelay	<p>Output delay. As an unsigned 32-bit value. Allowed values: = 0: Value is outputted only after <code>PreviewTime</code> has expired. > 0: Value is outputted immediately.</p>



Undelayed Short List Command	write_port_list
Comments	<ul style="list-style-type: none"> • NoDelay is only relevant, if a PreviewTime has been set by set_scanahead_params. • write_port_list(NoDelay = 0) behaves (depending on Port) similar to: <ul style="list-style-type: none"> – write_da_x_list – write_8bit_port_list – write_io_port_list – set_laser_pin_out_list
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version info	Available as of DLL 634 , OUT 635 , RBF 634 .
References	set_scanahead_params



10.3 Unsupported RTC4/RTC5 Commands

Some RTC4/RTC5 commands are not supported by the RTC6. Most of them can be replaced by RTC6 commands, see following tables.

Ctrl Command	dsp_start
Support status	This RTC4 command is not supported by the RTC6.
RTC4→RTC6	This command is not needed for normal operation. The DSP starts automatically after the program file is loaded by load_program_file .

Ctrl Command	free_RTC5_dll
Support status	This RTC5 command is not supported by the RTC6.
Replaced by	free_RTC6_dll

Ctrl Command	get_xy_pos
Support status	This RTC4 command is not supported by the RTC6.
RTC4→RTC6	Replaced by get_value, get_values

Ctrl Command	get_xyz_pos
Support status	This RTC4 command is not supported by the RTC6.
RTC4→RTC6	Replaced by get_value, get_values

Ctrl Command	init_RTC5_dll
Support status	This RTC5 command is not supported by the RTC6.
Replaced by	init_RTC6_dll

Ctrl Command	read_pixel_ad
Support status	This RTC4 command is not supported by the RTC6.
RTC4→RTC6	There is no equivalent RTC6 command.

Ctrl Command	rtc4_count_cards
Support status	This RTC4 command is not supported by the RTC6.
Replaced by	rtc6_count_cards

Ctrl Command	rtc5_count_cards
Support status	This RTC5 command is not supported by the RTC6.
Replaced by	rtc6_count_cards



Ctrl Command	select_list
Support status	This RTC4 command is not supported by the RTC6.
RTC4→RTC6	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 (for example, after a board changed "ownership") by set_start_list_2 and get_input_pointer .

Ctrl Command	set_list_mode
Support status	This RTC4 command is not supported by the RTC6.
RTC4→RTC6	See Chapter 6.5.4 "RTC4-Circular Queue Mode", page 120 .

Ctrl Command	set_piso_control
Support status	This RTC4 command is not supported by the RTC6.
RTC4→RTC6	<p>This command is not needed for operation of the RTC6.</p> <p>For data transfer in accordance with the SL2-100 protocol, the bi-directional communication between the scan system and the RTC5/RTC6 does not depend on the data cable length.</p> <p>For data transfer in accordance with the XY2-100 protocol, the timing of the communication must be further on adjusted to reflect the length of the data cable; but the adjustment is now realized by a jumper at the XY2-100 Converter (Accessory).</p>

Ctrl Command	set_wobbel_xy
Support status	This RTC4 command is not supported by the RTC6.
Replaced by	set_wobbel , set_wobbel_mode

Ctrl Command	z_out
Support status	This RTC4 command is not supported by the RTC6.
RTC4→RTC6	For setting the z value in a 3-axis scan system, set_defocus can be used (only together with an RTC6 with enabled Option "3D"). 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 RTC4 command is not supported by the RTC6.
RTC4→RTC6	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

- Currently the RTC6 Software Package does not contain demo programs.

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 RTC6 board is correctly seated in the PCIe slot. Refer to the instructions in your PC manual. Check for metal parts that may have fallen into the PC housing during installation. Check for loose cables or connectors.
RTC6 board does not respond	<ul style="list-style-type: none"> Check the driver installation. See Chapter 5.4 "Installing the RTC6 Software", page 89. 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 (Accessory), then check, whether the solder jumpers of the XY2-100 Converter (Accessory) are set appropriate for the cable length. See Figure 14. If yes: Check the RTC6 DLL import declarations in your user program. See Chapter 6.2.2 "Importing Commands", page 94.
User program fails	<ul style="list-style-type: none"> Check the driver installation. See Chapter 5.4 "Installing the RTC6 Software", page 89. Check the RTC6 board initialization in the user program. Check the RTC6 DLL import declarations in your user program. See Chapter 6.2.2 "Importing Commands", page 94.
Scan head control fails	<ul style="list-style-type: none"> Check if the scan head is properly connected to the RTC6 board by the data cable. Make sure to follow the specifications for the data cable. See Chapter 4.5.3 "Data Cables (Accessories)", page 70. 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 RTC6 board and the laser.
Irregular marking results	<ul style="list-style-type: none"> Check the Laser Delays and Scanner Delays. See Chapter 7.2.3 "Notes on Optimizing the Delays", page 154.
Laser does not switch off during Jump commands	<ul style="list-style-type: none"> Check the Laser Delays and Scanner Delays. See Chapter 7.2 "Delay Settings – Coordinating Scan Head Control and Laser Control", page 144.

Additionally, the following commands are 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 129](#).
- With `set_verify`, you can verify that all downloads (commands, tables) were performed error-free, see [Section "Download Verification", page 130](#).
- 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 return values, see [Section "Status Monitoring and Diagnostics", page 182](#) (for *iDIVE* scan systems see also [Chapter 8.1 "iDIVE Functions", page 211](#)).
- 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 cycle occurred, see [Section "Clock Overruns", page 181](#).
- With `get_status` or `get_out_pointer`, you can determine which command number the program is currently executing (for example, for "infinite loops" due to a circular argument in the program flow).
- `get_startstop_info` provides information about the laser control 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 [Section "Normal, Short, Variable and Multiple List Commands", page 301](#)) are typically used for this, one command might overwrite the other's output value within the same 10 µs clock cycle. In this situation, separate both short commands with a (normal) list command, for example, `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 cycle 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 154](#).

If the problems persist, contact SCANLAB.



13 Customer Service

13.1 Servicing and Repairs

All RTC6 board servicing and repairs should be performed only at SCANLAB. The warranty expires if the RTC6 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 RTC6 board repairs not performed by SCANLAB
- for damage resulting from improper packaging of a product returned to SCANLAB
- if the RTC6 board has been altered
- 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 RTC6 board can be returned to SCANLAB for a fee to be properly disposed of.



14 Legal

14.1 EU Declaration of Conformity – RTC6 PCIe Board


<p>EU-Konformitätserklärung</p> <p>für das Produkt:</p> <p>RTC4 (PCI); RTC4 PCI-Express; RTC5 (PCI); RTC5 PCI-Express; RTC6 PCI-Express</p> <p>Der Hersteller</p> <p>SCANLAB GmbH, Siemensstr. 2a, 82178 Puchheim, Deutschland</p> <p>erklärt, dass das genannte Produkt die einschlägigen Harmonisierungsrechtsvorschriften der Union erfüllt:</p> <ul style="list-style-type: none">• 2014/30/EU - Richtlinie des EUROPÄISCHEN PARLAMENTS UND DES RATES zur Harmonisierung der Rechtsvorschriften der Mitgliedstaaten über die elektromagnetische Verträglichkeit (EMV-Richtlinie).• 2011/65/EU - Richtlinie des EUROPÄISCHEN PARLAMENTS UND DES RATES zur Beschränkung der Verwendung bestimmter gefährlicher Stoffe in Elektro- und Elektronikgeräten (RoHS-Richtlinie). <p>Folgende harmonisierte Normen wurden angewandt:</p> <ul style="list-style-type: none">• EN IEC 61000-6-2:2005 – Elektromagnetische Verträglichkeit (EMV) - Teil 6-2: Fachgrundnormen - Störfestigkeit für Industriebereiche• DIN EN 55011:2018-05 - Industrielle, wissenschaftliche und medizinische Geräte - Funkstörungen - Grenzwerte und Messverfahren• DIN EN IEC 63000:2019-05 - Technische Dokumentation zur Beurteilung von Elektro- und Elektronikgeräten hinsichtlich der Beschränkung gefährlicher Stoffe <p>Die alleinige Verantwortung für die Ausstellung dieser Konformitätserklärung trägt der Hersteller.</p> <p>Dieses Dokument zur Kundeninformation wurde elektronisch ausgestellt und ist daher ohne Unterschrift gültig. Eine unterschriebene Ausführung ist Bestandteil der SCANLAB-internen technischen Dokumentation.</p> <p><i>EU Declaration of Conformity</i></p> <p>for the product:</p> <p>RTC4 (PCI); RTC4 PCI-Express; RTC5 (PCI); RTC5 PCI-Express; RTC6 PCI-Express</p> <p>The manufacturer</p> <p>SCANLAB GmbH, Siemensstr. 2a, 82178 Puchheim, Germany</p> <p>declares that the product mentioned above complies with the relevant Union harmonization legislation:</p> <ul style="list-style-type: none">• 2014/30/EU - Directive of the EUROPEAN PARLIAMENT AND OF THE COUNCIL on the harmonisation of the laws of the Member States relating to electromagnetic compatibility (EMC Directive)• 2011/65/EU - Directive of the EUROPEAN PARLIAMENT AND OF THE COUNCIL on the restriction of the use of certain hazardous substances in electrical and electronic equipment (RoHS Directive). <p>The following harmonized standards have been applied:</p> <ul style="list-style-type: none">• EN IEC 61000-6-2:2005 - Electromagnetic compatibility (EMC) - Part 6-2: Generic standards - Immunity for industrial environments• DIN EN 55011:2018-05 - Industrial, scientific and medical equipment – Radio-frequency disturbance characteristics - Limits and methods of measurement• DIN EN IEC 63000:2019-05 - Technical documentation for the assessment of electrical and electronic products with respect to the restriction of hazardous substances <p>This declaration of conformity is issued under the sole responsibility of the manufacturer.</p> <p>This document, used for customer information, was issued electronically and is therefore valid without a signature. A signed version is part of the SCANLAB internal technical documentation.</p> <p>SCANLAB GmbH Puchheim 2021-07-19</p> <p>© SCANLAB GmbH – 2021/07</p>



14.2 Compliance with FCC Rules

The RTC6 PCIe 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 RTC6 PCIe Board is operated in a commercial environment. The RTC6 PCIe 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 RTC6 PCIe 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 Technical Specifications – RTC6 PCIe Board

System Requirements		Interfaces	
Windows PC with free PCI-Express slot			
Operating system	With RTC6 Software Package ≥ V1.3.0 : Microsoft Windows 10, 8, 7 as 32-bit or 64-bit version. <i>Windows XP and Windows Vista are not supported!</i>	SCANHEAD Connector Connector	9-pin D-SUB connector (female)
Dimensions		2. SCANHEAD Socket Connector	
Length	160 mm	Connector	10-pin socket connector.
Height	104.5 mm		<ul style="list-style-type: none"> xy output values only with enabled Option "Second Scan Head Control" z output values only with enabled Option "3D"
Interface to the PC		Signal transmission	SL2-100 protocol.
PCI-Express x-1, version 1.0			Via XY2-100 Converter (Accessory) : XY2-100 protocol.
Scan System Control			
Number of list memory	Up to 3, configurable areas		
Total capacity of list memory	8,388,608 list positions		
Output interval of Microsteps	10 μ s		
Maximum range for the Image field coordinates (20-bit signed)	–524,288...+524,287		
Virtual Image field (for example, for Processing-on-the-fly)	–268,435,456... +268,435,455 (28 bit, signed = 29 bit) ^(a)		

(a) With RTC6 Software Package \geq 1.4.0.

LASER Connector

Connector	15-pin D-SUB connector (female)	Input ports for external start and stop signals	TTL active-LOW, internally connected to +3.3 V by pull-up resistors (4.7 kΩ)
laser control signals	LASER1, LASER2, LASERON	• /START	edge sensitive HIGH level > 2.3 V LOW level < 0.6 V
• TTL level	5 V, active-HIGH or active-LOW (programmable)	• /STOP	level sensitive HIGH level > 2.3 V LOW level < 0.6 V
• Max. current load	20 mA	• Reference	GND ^(a)
• Reference	GND ^(a) (GND2 with Option "DC/DC Converter")	Other signals	
Analog outputs	ANALOG OUT1, ANALOG OUT2	• BUSY OUT	5 V, TTL active-HIGH, max. 10 mA
• Output voltage range	0 V...10 V	• +5 V	max. 100 mA
• Resolution	12 Bit	• Reference	GND ^(a)
• Max. current load	5 mA		
• Reference	GND ^(a)		
Digital input port	2 Bits		
• 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		
• Reference	GND ^(a)		
Digital output	2 bits, buffered		
• LOW level	< 0.55 V		
• HIGH level	> 3.8 V		
• Max. current load	20 mA		
• Reference	GND ^(a)		

(a) See footnote on [page 72](#).

EXTENSION 1 Socket Connector

Connector	40-pin socket connector, output signal level configurable by jumper
Digital output	16 bits, buffered
• LOW level	< 0.4 V
• HIGH level	> 2.0 V (3.3 V or 5 V)
• Max. current load	±8 mA
• Reference	GND ^(a)
• LATCH signal	3.3 V or 5 V, TTL active-HIGH, 5 µs pulse, max. 10 mA
Digital input port	16 bits
• 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 ^(a)
• SYNC signal	3.3 V or 5 V, TTL active-HIGH, square wave signal (5 µs pulse, 10 µs period) max. 10 mA
Other signals	
• BUSY OUT	3.3 V or 5 V, TTL active-HIGH, max. 10 mA
• VCC	3.3 V or 5 V, max. 100 mA
• +5 V	max. 100 mA
• Reference	GND ^(a)

EXTENSION 2 Socket Connector

Connector	26-pin socket connector, configurable by jumper
Digital output	8 bits, buffered
• LOW level	< 0.4 V
• HIGH level	> 2.0 V
• Max. current load	±8 mA
• Reference	GND ^(a)
• LATCH signal	5 V, TTL active-HIGH, 5 µs pulse, max. 10 mA
Laser control signals	LASERON, LASER1, LASER2 (see LASER Connector , page 855)
Other signals	
• +5 V	max. 100 mA
• Reference	GND ^(a)

MARKING ON THE FLY Socket Connector

Connector	16-pin socket connector
2 encoder input ports for incremental encoders	ENCODER X (1±,2±) and ENCODER Y (1±,2±), designed for a pair of standardized differential input signals (RS-422) each. HIGH level ≥ 2.0 V LOW level ≤ 0.8 V f ≤ 4 MHz
Analog outputs	ANALOG OUT2 (see LASER Connector)
Input ports for external start and stop signals	/START2, /STOP2 (see /START, /STOP of the LASER Connector)
Other signals	
• BUSY OUT	5 V, TTL active-HIGH, max. 10 mA
• +5 V	max. 100 mA
• Reference	GND ^(a)

RS232 Socket Connector

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 ^(a)
Baud rate	300...115200

STEPPER MOTOR Socket Connector

Connector	10-pin socket connector
Signals for controlling two stepper motors:	
• ENABLE	5 V, TTL output port,
	DIRECTION output port
• CLOCK output port	5 V, TTL active-HIGH, 5 μ s pulse
• SWITCH input port	TTL active-LOW, internally connected to +3.3 V by pull-up resistors (10 k Ω)

McBSP/ANALOG Socket Connector

Connector	10-pin socket connector
McBSP interface	See Section "McBSP Interface", page 83.
	• 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 Data delay XDelay bits Data delay RDelay bits
• Reference	GND ^(a)
SPI interface functionality	No
Analog input ports	ANALOG IN0, ANALOG IN1
	• Input voltage range 0 V...10 V
	• Input resistance > 5 k Ω
	• ADC resolution 12 bit
• Reference	GND ^(a)

16 Appendix A: The RTC6 Ethernet Board

16.1 Product Overview

16.1.1 RTC6 Ethernet Board vs. RTC6 PCIe Board – Usage and Comparison

RTC6 Ethernet Boards and RTC6 PCIe Boards share the same *core functionality*:

- Real-time control of the laser and scan system.
- Transfer of direct commands (control commands) from the computer to the board. These commands are executed immediately.
- Transfer of list commands from the computer to the list memory of the board. These commands execute only after the list is started.
- List execution occurs in real time.
- Calculation of the set position occurs every 10 µs.
- **Image field** correction is applied to the set positions.
- The interface to the scan system uses the 20-bit SL2-100 protocol.
- For software development, the total command set for RTC6 Ethernet Boards and RTC6 PCIe Boards is contained in [RTC6DLL.dll/RTC6DLLx64.dll](#).

RTC6 Ethernet Boards differ in the following aspects:

- Ethernet interface⁽¹⁾ instead of PCIe bus
 - Communication with the computer is by TCP/IP and UDP.
 - RTC6 Ethernet Boards work *without* RTC6 board driver.
 - The board does not need to be plugged into a PC.
 - Real-time clock, see [Chapter 16.2.15 "Real-Time Clock", page 879](#).
 - With RTC6 Software Package ≥ V1.7.0, RTC6 Ethernet Boards can also be used in standalone operation without a PC, see [Chapter 16.7 "Standalone Functionality", page 890](#).

(1) Supports Gigabit Ethernet according to 1000BASE-T.

16.1.2 System Requirements



Caution!

- To avoid interference coupling and emissions, never operate the RTC6 Ethernet Board without shielding (e.g. outside a metal housing).
- The RTC6 Ethernet Board is designed exclusively for industrial use. It is intended for integration in a machine (typically in a laser system) and does **not** fulfill all requirements of a ready-to-use consumer end product. Only operate the RTC6 Ethernet Board after integration in a machine meeting all applicable directives and standards (of your local jurisdiction). It is *not* suitable for use as a toy, in households or inclement environments (e.g. outdoors). The operator must take appropriate precautionary measures to prevent such improper usage.
- Installation and commissioning should only be performed by personnel with sufficient qualifications, e.g. knowledge of electrical equipment safety. Only perform installation and maintenance if power and lasers are switched off.
- Because the RTC6 Ethernet Board is a Class A device capable of generating interference in residential areas, the operator may be required to carry out appropriate measures.

Hardware

- Shielded housing with:
 - Provisions for mechanical mounting, incl. mounting hardware
 - appropriate thermal coupling (cooling) of the board
 - Power outlet
 - Ethernet cable
 - Power supply as specified
 - Cable with connector for powering the RTC6 Ethernet Board
 - Flat ribbon cables for peripheral equipment (laser, scan head, extensions)
- Windows PC
- Gigabit Ethernet (preferred), 10/100 Mbit/s Ethernet

Software

- TCP/IP protocol IPv4
- UDP protocol
- Used ports (default)
 - 63749 (UDP)
 - 63750 (UDP, TCP)

For the development of user programs:

- DLL/Import declarations (contained in RTC6 Software Package)

16.1.3 Options

Same as RTC6 PCIe Boards, see [Chapter 2.6 "Options", page 37](#). Additionally, with RTC6 Ethernet Boards only:

- **Option "LDSA"**
 - **Support of Standalone Functionality by laserDESK**
Allows **Standalone Functionality**-related features to be used in laserDESK.
Example: Booting an RTC6 Ethernet Board with a **Boot Image** created under laserDESK.
Note that **Boot Images** created under laserDESK cannot be used with RTC6 Ethernet Boards without **Option "LDSA"**. For more information, refer to the laserDESK online help.
See also [get_RTC_version](#), Bit #11.

16.1.4 Labeling

Same as RTC6 PCIe Boards, see [Chapter 2.1 "Labeling", page 30](#).

16.1.5 Type Identification

Same as RTC6 PCIe Boards, see [Chapter 2.7 "Jumper Settings and Type Designations", page 39](#).

16.1.6 Unpacking Instructions and Typical Scope of Delivery

Same as RTC6 PCIe Boards, see [Chapter 2.2 "Unpacking Instructions and Typical Scope of Delivery", page 30](#).

16.1.7 Delivered Software

Same as RTC6 PCIe Boards, see [Chapter 2.3 "Delivered RTC6 Software Package", page 30](#).

16.1.8 Accessories for the RTC6 PCIe Board

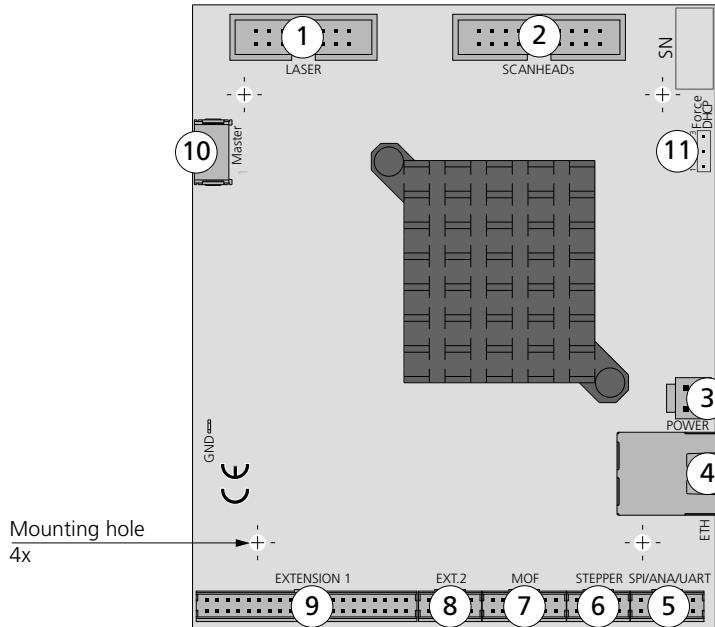
Accessories mentioned in [Chapter 2.8 "Accessories for the RTC6 PCIe Board", page 43](#) for RTC6 PCIe Board is (without suiting adapters) not suitable for RTC6 Ethernet Boards (because of different connector plugs and pitch of the pins).

16.1.9 Supplementary Software

As with RTC6 PCIe Boards, see [Chapter 2.9 "Supplementary Software", page 46](#).

16.2 Layout, Interfaces, Jumper Settings

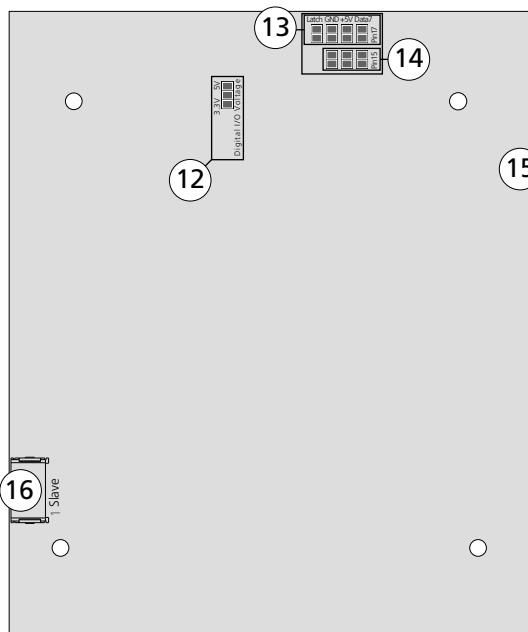
16.2.1 Layout – Upper Side



Legend

1. LASER 16-pin socket connector. To connect the laser. With digital and analog outputs. For details, see "[LASER Socket Connector](#)", page 864.
2. SCANHEADS 20-pin socket connector. To connect the first and second scan head. For details, see "[SCANHEADS Socket Connector](#)", page 867.
3. POWER 2-pin socket connector. For the voltage supply. For details, see "[POWER Socket Connector](#)", page 869.
4. ETH RJ-48 8P8C connector. For the network connection. For details, see [Chapter 16.2.7 "ETH Connector"](#), page 869.
5. SPI/ANA/UART 12-pin socket connector. Hardware interface for several data exchange methods. With analog input ports. For details, see "[SPI/ANA/UART Socket Connector](#)", page 870.
6. STEPPER 10-pin socket connector. For controlling up to two stepper motors. For details, see "[STEPPER Socket Connector](#)", page 871.
7. MOF 14-pin socket connector. Hardware interface for encoder pulses, for example, for Processing-on-the-fly applications. For details, see "[MARKING ON THE FLY Socket Connector](#)", page 81.
8. EXT. 2 10-pin socket connector. With a 8-bit digital output port. For details, see "[EXTENSION 2 Socket Connector](#)", page 79.
9. EXTENSION 1 40-pin socket connector. With a 16-Bit Digital Output and a 16-Bit digital input port. For details, see "[EXTENSION 1 Socket Connector](#)", page 77.
10. Master 6-pin socket connector. To connect with another RTC6 board for the purpose of synchronize clocks. For details, see "[Master Socket Connector, Slave Socket Connector](#)", page 875.
11. Force DHCP Jumper field Jumper field to ignore or not to ignore the saved static IP address. For details, see "[Jumper Field 'Force DHCP'](#)", page 879.

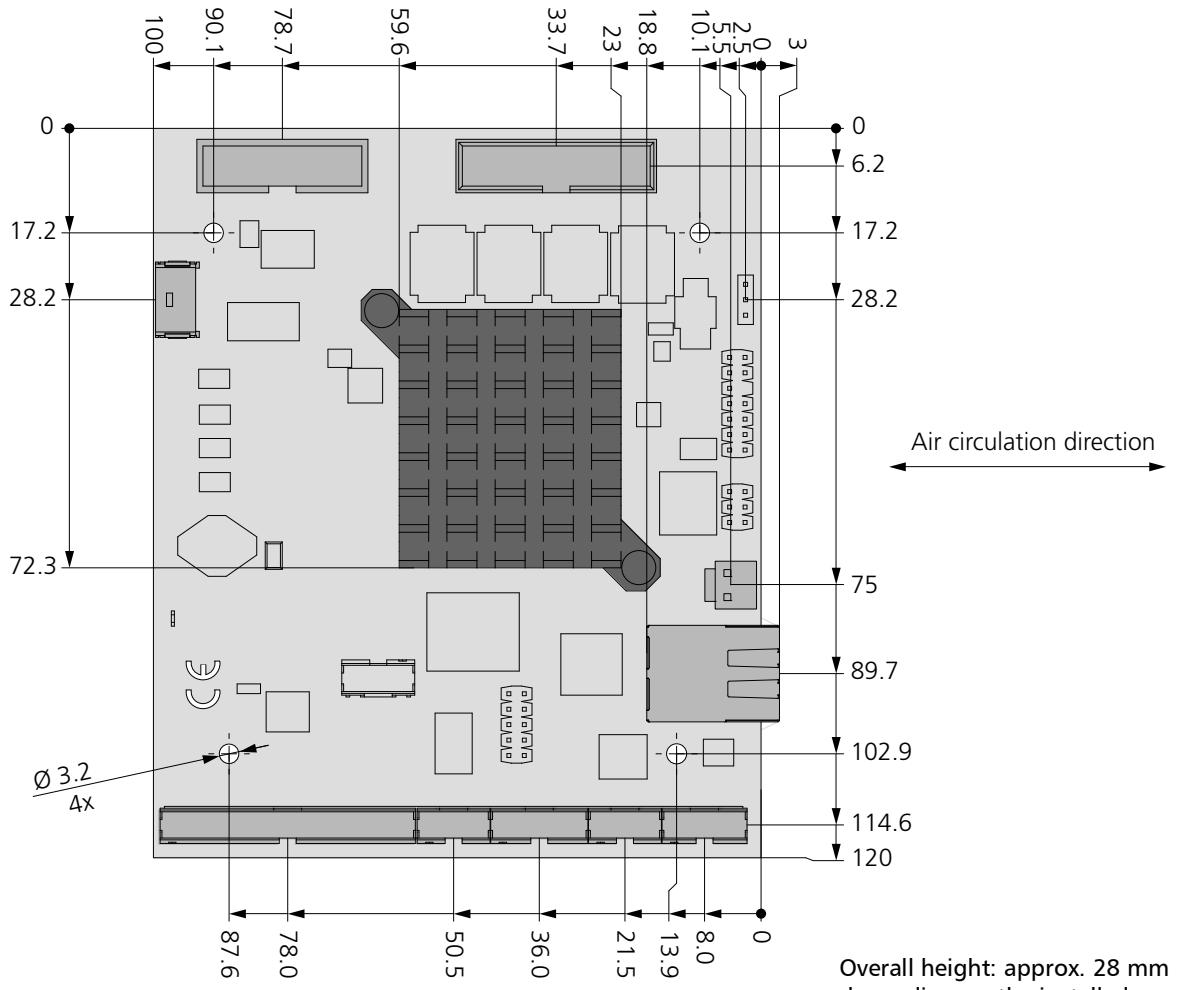
16.2.2 Layout – Lower Side



Legend

- 12. Solder jumper field A To configure the level of the output signals at the EXTENSION 1 socket connector.
For details, see "[Solder Jumper Field A – Configuring the Output Signal Level at EXTENSION 1 Socket Connector](#)", page 40.
- 13. Solder jumper field B To configure the signal at EXT. 2 socket connector pin (09).
For details, see "[Solder Jumper Field B – Configuring pin \(09\) of EXT. 2 Socket Connector](#)", page 877.
- 14. Solder jumper field C To configure the signal at EXT. 2 socket connector pin (08).
For details, see "[Solder Jumper Field C – Configuring Pin \(15\) of EXTENSION 2 Socket Connector](#)", page 42.
- 15. ETH connector See [Figure 72](#).
- 16. Slave. 6-pin socket connector. To connect with another RTC6 board for the purpose of synchronize clocks. For details, see "[Master Socket Connector, Slave Socket Connector](#)", page 875.

16.2.3 Dimensions and Connector Positions

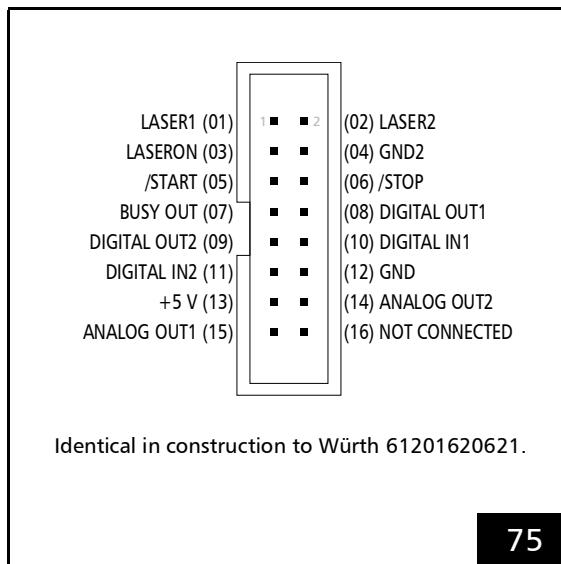


RTC6 Ethernet Board V1.2. Dimensions and connector positions. All dimensions in mm.

16.2.4 LASER Socket Connector

The LASER socket connector has 16 pins⁽¹⁾. It is located on the upper side of the RTC6 Ethernet Board, see [Figure 72](#), number 1.

The pin-out is shown in [Figure 75](#).



75

LASER socket connector: pin-out. The pitch of the pins is 2.54 mm. On RTC6 Ethernet Boards without [Option "DC/DC Converter"](#), GND and GND2 are identical. On RTC6 Ethernet Boards with [Option "DC/DC Converter"](#), GND2 and the laser output signal are galvanically decoupled from GND.

Notice!

- If you want to use the RTC6 Ethernet Board in conjunction with laserDESK, observe the following:
 - laserDESK uses additional pins on the [EXTENSION 1 Socket Connector](#), for example, to control the laser.
 - Refer to the extended documentation for the LASER connector which can be found in the laserDESK online help (alternatively available from SCANLAB or in laserDESK-zip, which can be downloaded from the SCANLAB website).

(1) With RTC6 PCIe Boards: 15 pin D-SUB connector, female.

Notes

- With RTC6 Ethernet Boards, GND is the ground at the POWER connector. See [Chapter 16.2.6 "POWER Socket Connector", page 869](#). With the RTC6 PCIe Board, GND is the PC ground.
- A suitable 0.2 m cable (#116048) is available from SCANLAB, see also [Figure 76](#).

Laser Control Signals

As with RTC6 PCIe Boards, see [Chapter 4.6.1 "LASER Connector", page 72](#). However, note that the pin numbers differ.

Signal	RTC6 Ethernet Board	RTC6 PCIe Board
LASERON	Pin (03)	Pin (02)
LASER1	Pin (01)	Pin (01)
LASER2	Pin (02)	Pin (09)

External Control Signals

As with RTC6 PCIe Boards, see [Chapter 4.6.1 "LASER Connector", page 72](#).

BUSY List Execution Status

As with RTC6 PCIe Boards, see [Chapter 4.6.1 "LASER Connector", page 72](#). However, note that the pin numbers differ.

Signal	RTC6 Ethernet Board	RTC6 PCIe Board
BUSY OUT	Pin (07)	Pin (04)

2-Bit Digital Input Port and 2-Bit Digital Output Port

As with RTC6 PCIe Boards, see [Chapter 4.6.1 "LASER Connector", page 72](#). However, note that the pin numbers differ.

Signal	RTC6 Ethernet Board	RTC6 PCIe Board
DIGITAL IN1	Pin (10)	Pin (13)
DIGITAL IN2	Pin (11)	Pin (06)
DIGITAL OUT1	Pin (08)	Pin (12)
DIGITAL OUT2	Pin (09)	Pin (05)

12-Bit Analog Output Ports

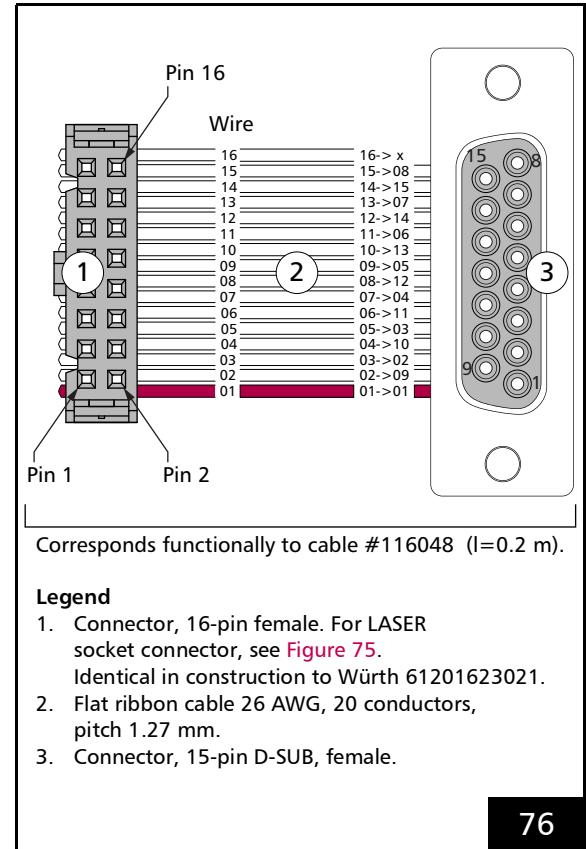
As with RTC6 PCIe Boards, see [Chapter 4.6.1 "LASER Connector", page 72](#). However, note that the pin numbers differ.

Signal	RTC6 Ethernet Board	RTC6 PCIe Board
ANALOG OUT1	Pin (15)	Pin (08)
ANALOG OUT2 ^(a)	Pin (14)	Pin (15)

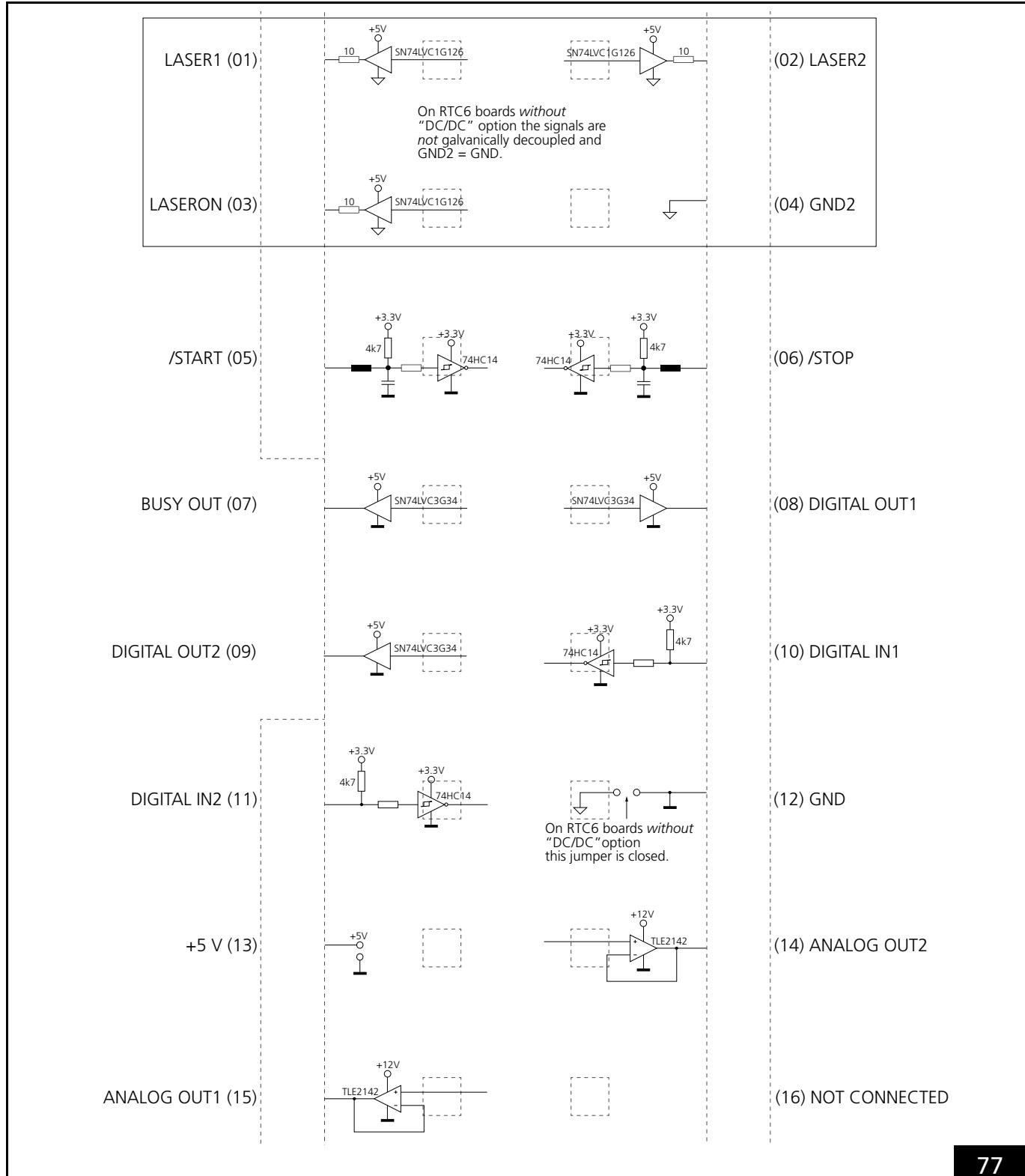
(a) With the RTC6 PCIe Board the ANALOG OUT2 signal is additionally available at the MARKING ON THE FLY socket connector.

Input and Output Wiring

The input and output wiring of the 16-pin LASER socket connector is shown in [Figure 77](#). It is identical to RTC6 PCIe Boards, except the pin numbers.



Cable (proposal). Actual implementation may differ according to customer needs.
The depicted items are not in the standard scope of delivery of the RTC6 Ethernet Board!



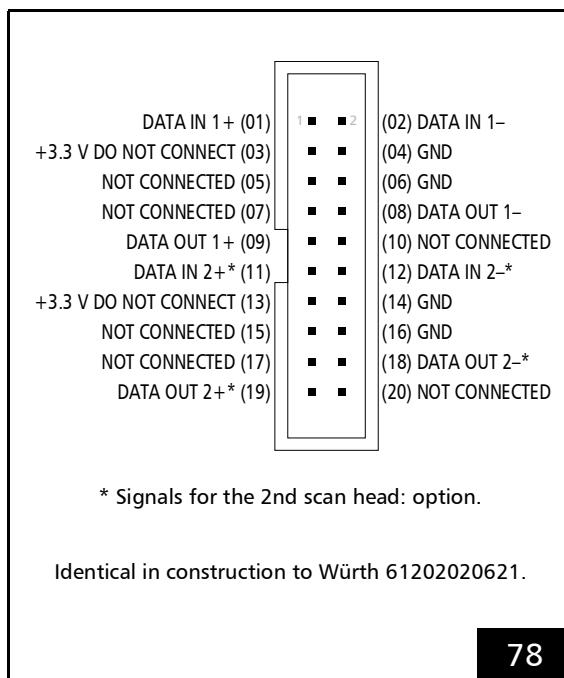
LASER socket connector (detail, indicated by dashed line): input/output wiring plan.

16.2.5 SCANHEADs Socket Connector

The SCANHEADs socket connector has 20 pins. It is located on the upper side of the RTC6 Ethernet Board, see [Figure 72](#), number 2.

The SCANHEADs socket connector provides the same signals as the SCANHEAD and 2. SCANHEAD socket connector of the RTC6 PCIe Board for the first scan head connector (pin (01)...pin (10)) and the second scan head-connector (pin (11)...pin (20)).

The pin-out is shown in [Figure 78](#).



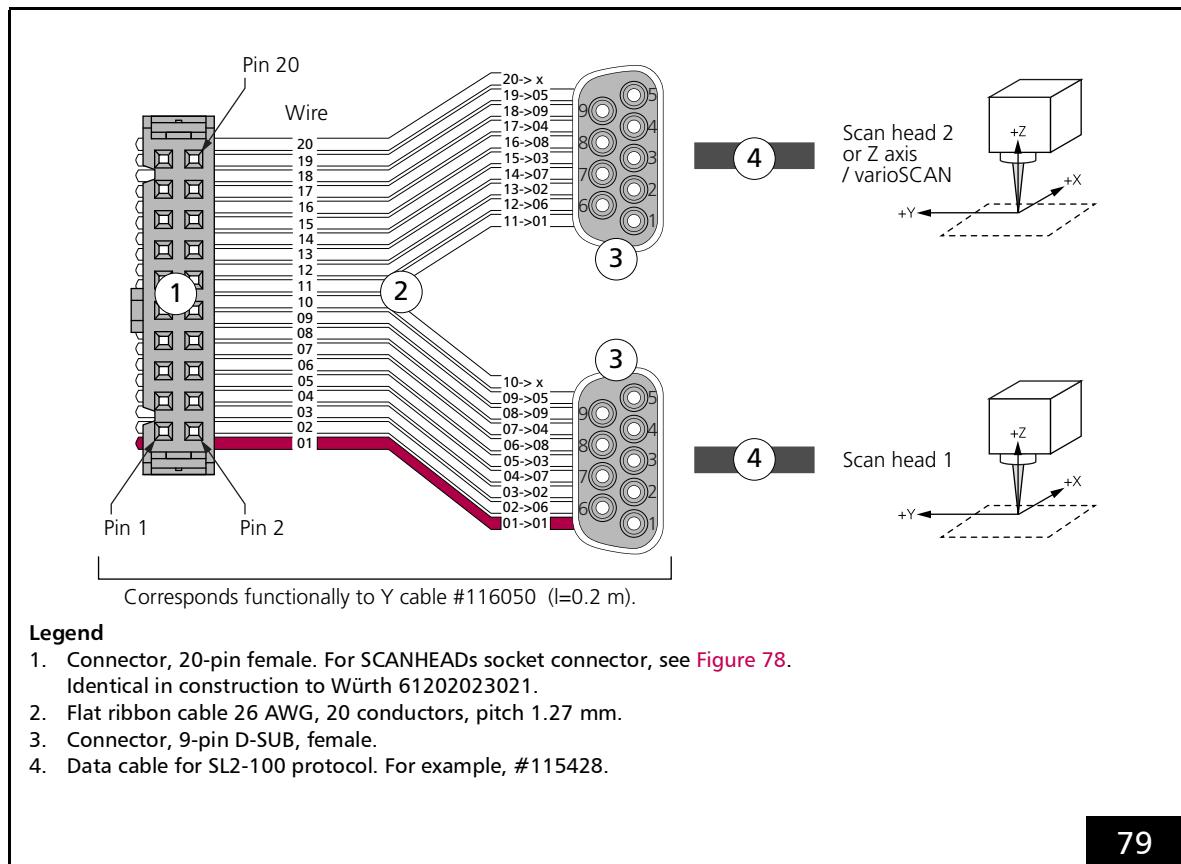
78

SCANHEADs socket connector: pin-out. The pitch of the pins is 2.54 mm.

A cabling example is shown in [Figure 79](#) (the shown Y cable is available from SCANLAB with a length of 0.2 m: #116050).

Scan system control (via an [XY2-100 Converter \(Accessory\)](#), if needed) is identical to that of the RTC6 PCIe Board.

For further information, see [Chapter 4.5 "Interfaces to Scan System", page 66](#).

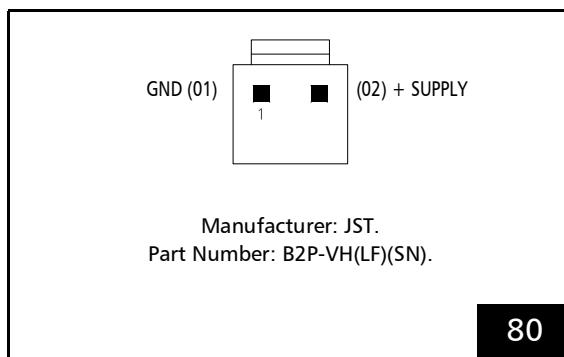


Cabling (proposal). Actual implementation may differ according to customer needs.
The depicted items are not in the standard scope of delivery of the RTC6 Ethernet Board!

16.2.6 POWER Socket Connector

The POWER socket connector has 2 pins. It is located on the upper side of the RTC6 Ethernet Board, see [Figure 72](#), number **3**. It serves to provide the voltage supply for the RTC6 Ethernet Board.

The pin-out is shown in [Figure 80](#).



80

POWER socket connector. The pitch of the pins is 3.96 mm.

Pin	Power supply
(01) GND	Ground.
(02) + Supply	See Chapter 16.10 "Technical Specifications – RTC6 Ethernet Board", page 900 .

Notes

- Required parts for the mating connector are *not included in the scope of delivery*:

Manufacturer: JST

1× Housing, part number VHR-2N

2× Contact, part number SVH-41T-P1.1.

16.2.7 ETH Connector

The ETH connector is an 8P8C type RJ connector (colloquially: RJ-45). It is located on the upper side of the RTC6 Ethernet Board, see [Figure 72](#), number **4**. It serves to connect⁽¹⁾ the RTC6 Ethernet Board to the network.

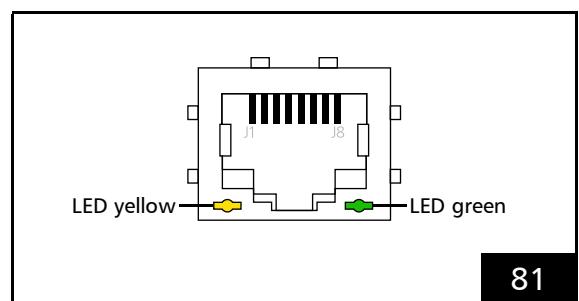
For status indication the following LEDs are part of the connector, see [Figure 81](#):

- LED yellow, function "IP address assigned". The LED is *permanently on* as soon as the RTC6 Ethernet Board has an assigned IP address (static IP, DHCP or as of [BIOS](#) 24 link-local address⁽²⁾).

Note: Without stored static IP address the switched-on LED indicates that the IP address assignment has been carried-out successfully by DHCP. In case if there is a stored static IP address a switched-on LED does not mean that the IP address is valid in the subnet in every case. As soon as the board has been acquired by a PC the LED *flashes* when there is network traffic ("Traffic").

- LED green, function "Link / Traffic". The LED is *permanently on* as soon as a connection is established ("Link"). This is usually the case when the Ethernet cable is plugged-in and there is an active router or a LAN adapter at the remote end.

The LED *flashes* when there is network traffic ("Traffic").



81

ETH connector. RJ-45 8P8C.

(1) Category 5e (Cat 5e) Ethernet cable recommended.

(2) See [page 881](#).

16.2.8 SPI/ANA/UART Socket Connector

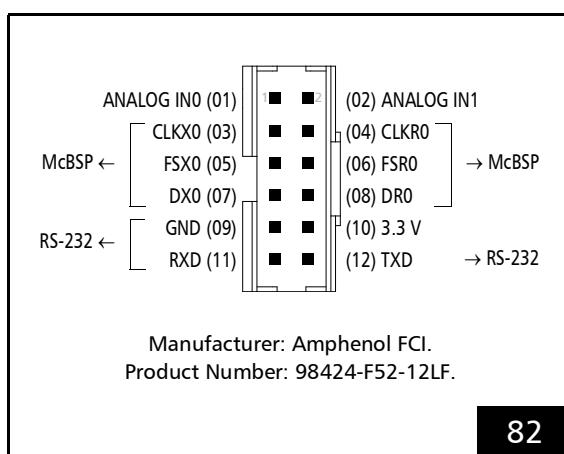
The SPI/ANA/UART socket connector has 12 pins. It is located on the upper side of the RTC6 Ethernet Board, see [Figure 72](#), number 5.

The SPI/ANA/UART socket connector is a hardware interface intended for analog input ports as well as for the data exchange methods **McBSP** (Multichannel Buffered Serial Port) and RS-232.

The SPI/ANA/UART socket connector corresponds functionally to the:

- McBSP/ANALOG socket connector and RS232 socket connector of RTC6 PCIe Boards

The relevant pin-outs are shown in [Figure 82](#).



Analog Input Ports

The SPI/ANA/UART socket connector provides two analog input ports:

- **ANALOG IN0**
 - See [Section "Analog Input Ports", page 85](#)
- **ANALOG IN1**
 - See [Section "Analog Input Ports", page 85](#)

For technical specifications, see [SPI/ANA/UART Socket Connector, page 904](#).

McBSP Interface

See [Section "McBSP Interface", page 83](#).

RS-232 Interface

With RTC6 Ethernet Boards, the RS232 socket connector pins of the RTC6 PCIe Board are integrated to the SPI/ANA/UART⁽¹⁾ socket connector.

Specifications and further information as with the RTC6 PCIe Board, see [Chapter 4.6.5 "RS232 Socket Connector", page 82](#).

The signals are referenced to GND, see [Notes, page 864](#).

82

SPI/ANA/UART socket connector: pin-out. The pitch of the pins is 2.00 mm.

Notes

- Required parts for the mating connector are *not included in the scope of delivery*:
Manufacturer: Amphenol FCI
Series. Minitek®
1 x Housing, part number 90311-012LF
12 x Contact, part number 77138-101LF.

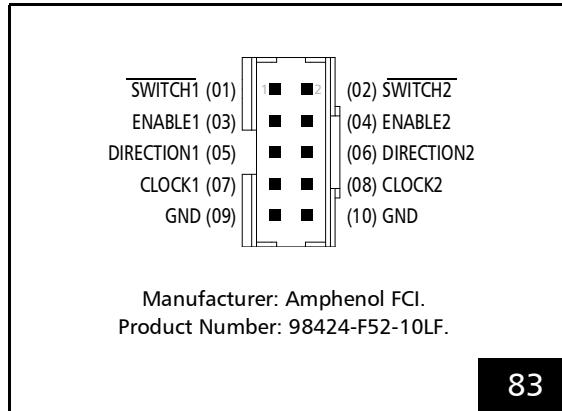
(1) The socket connector is labeled ".../UART" (Universal asynchronous receiver/transmitter). Refers to the electronic circuit which generates the data bits and the necessary data frame to be transferred on the RS-232 interface.

16.2.9 STEPPER Socket Connector

The STEPPER socket connector has 10 pins. It is located on the upper side of the RTC6 Ethernet Board, see [Figure 72](#), number 6.

At the STEPPER socket connector signals for controlling up to two stepper motors can be outputted.

The pin-out is shown in [Figure 83](#).



83

STEPPER socket connector: pin-out. The pitch of the pins is 2.00 mm.

All signals are referenced to GND, see [Notes, page 864](#).

Specifications and further information as with the RTC6 PCIe Board, [Chapter 4.6.7 "STEPPER MOTOR Socket Connector", page 86](#).

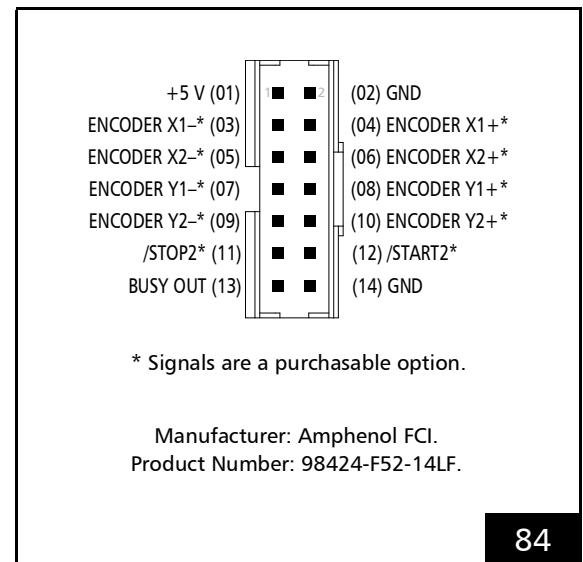
Notes

- Required parts for the mating connector are *not included in the scope of delivery*:
Manufacturer: Amphenol FCI
Series. Minitek®
1 × Housing, part number 90311-010LF
10 × Contact, part number 77138-101LF.

16.2.10 MOF Socket Connector

The MOF⁽¹⁾ socket connector has 14 pins. It is located on the upper side of the RTC6 Ethernet Board, see [Figure 72](#), number 7.

The pin-out is shown in [Figure 84](#).



84

MOF socket connector: pin-out. The pitch of the pins is 2.00 mm.

Notes

- Required parts for the mating connector are *not included in the scope of delivery*:
Manufacturer: Amphenol FCI
Series. Minitek®
1 × Housing, part number 90311-014LF
14 × Contact, part number 77138-101LF.
- RTC6 Ethernet Boards *do not* have a pin for the ANALOG OUT2 signal at their MOF socket connector, see [Chapter 4.6.4 "MARKING ON THE FLY Socket Connector", page 81](#).

(1) Marking on the Fly (= Processing-on-the-fly).

Encoder Input Ports

As with RTC6 PCIe Board, see [Section "Encoder Input Ports", page 81](#).

External Control Signals

As with RTC6 PCIe Board, see [Section "External Control Signals", page 81](#).

The signals are referenced to GND, see [Notes, page 864](#).

BUSY OUT Status

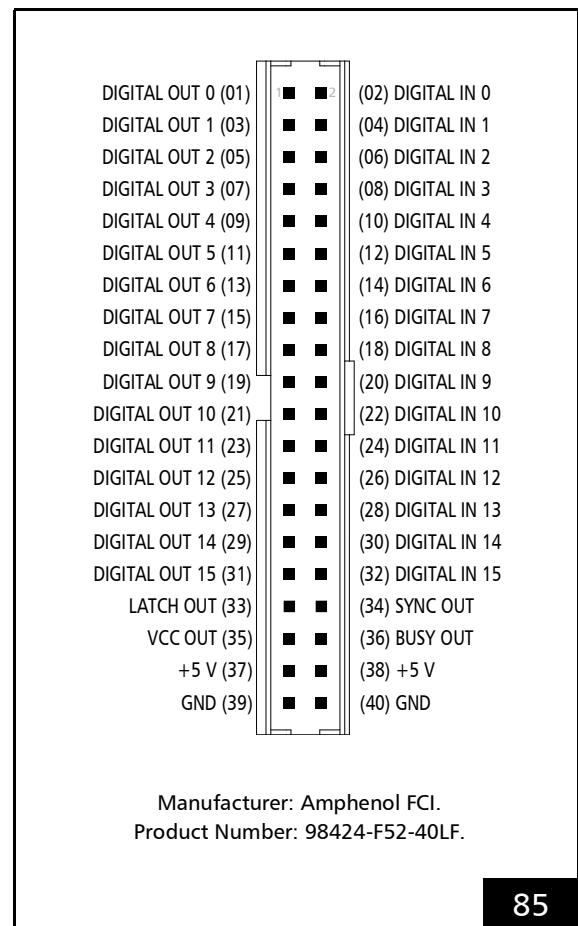
The BUSY OUT signal at pin (13) is identical to the BUSY OUT signal at the LASER socket connector, see [Chapter 16.2.4 "LASER Socket Connector", page 864](#).

The signal is referenced to GND, see [Notes, page 864](#).

16.2.11 EXTENSION 1 Socket Connector

The EXTENSION 1 socket connector has 40 pins. It is located on the upper side of the RTC6 Ethernet Board, see [Figure 72](#), number 9.

The pin-out is shown in [Figure 85](#).



85

EXTENSION 1 socket connector: pin-out. The pitch of the pins is 2.00 mm.

Notes

- Required parts for the mating connector are *not included in the scope of delivery*:
Manufacturer: Amphenol FCI
Series: Minitek®
1 x Housing, part number 90311-040LF
40 x Contact, part number 77138-101LF.
- The EXTENSION 1 socket connector on RTC6 Ethernet Boards has a 2.00 mm pitch of the pins, whereas RTC6 PCIe Boards have 2.54 mm.

Configuring the Output Signal Level

With the solder jumper field A on the lower side of the RTC6 Ethernet Board, the level of all output signals at the EXTENSION 1 socket connector (DIGITAL OUT 0...DIGITAL OUT 15, LATCH_OUT, SYNC_OUT, BUSY_OUT, VCC_OUT) can be configured for 5 V or 3.3 V, see [Chapter 2.7.1](#)

["Solder Jumper Field A – Configuring the Output Signal Level at EXTENSION 1 Socket Connector", page 40.](#)

The configured signal level is stationary outputted at pin (35): signal VCC_OUT. VCC_OUT is referenced to GND, see [Notes, page 864](#).

The maximum current load of the signal is 100 mA.

16-Bit Digital Output Port and 16-Bit Digital Input Port

As with RTC6 PCIe Board, see [Section "16-Bit Digital Input Port and 16-Bit Digital Output Port", page 77.](#)

The signals are referenced to GND, see [Notes, page 864](#).

Synchronization of Data Acquisition

As with RTC6 PCIe Board, see [Section "Synchronization of Data Acquisition", page 78.](#)

The signals are referenced to GND, see [Notes, page 864.](#)

BUSY List Execution Status

The BUSY OUT signal at pin (36) is identical to the BUSY OUT signal at the LASER socket connector, see [Chapter 16.2.4 "LASER Socket Connector", page 864.](#)

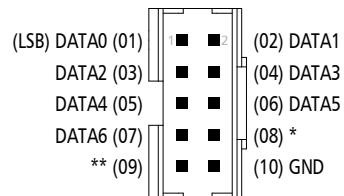
The signal is referenced to GND, see [Notes, page 864.](#)

16.2.12 EXT. 2 Socket Connector

The EXT. 2 socket connector has 10 pins. It is located on the upper side of the RTC6 PCIe Board, see [Figure 72, number 8.](#)

It provides a buffered 8-bit digital output port ((DATA0...DATA7).

The pin-out is shown in [Figure 24.](#)



* Jumper setting-dependent:
+5V or DATA7 or GND.

** Jumper setting-dependent:
+5V or DATA7 or GND or LATCH.

Manufacturer: Amphenol FCI.
Product Number: 98424-F52-10LF.

86

EXT. 2 socket connector: pin-out. The pitch of the pins is 2.00 mm.

On RTC6 PCIe Boards the designation is EXTENSION 2.

Apart from the total number of pins, the provided signals differ as well as the pin numbers, see following table.

Signal	RTC6 Ethernet Boards, EXT. 2 socket connect or 10 pins	RTC6 PCIe Boards, EXTENSION 2, 26 pins
DATA0	Pin (01)	Pin (01)
DATA1	Pin (02)	Pin (03)
DATA2	Pin (03)	Pin (05)
DATA3	Pin (04)	Pin (07)
DATA4	Pin (05)	Pin (09)
DATA5	Pin (06)	Pin (11)
DATA6	Pin (07)	Pin (13)
*, see in Figure 24	Pin (08)	Pin (15)
**, see in Figure 24	Pin (09)	Pin (17)
GND	Pin (10)	Pin (02)
+5 V	–	Pin (06), (18), (25)
LASER1	–	Pin (22)
LASER2	–	Pin (19)
GND2	–	Pin (23)

Notes

- Required parts for the mating connector are *not included in the scope of delivery*:
Manufacturer: Amphenol FCI
Series. Minitek®
1× Housing, part number 90311-010LF
10× Contact, part number 77138-101LF.
- The pin (08) and pin (09) are configurable by solder jumpers.
- RTC6 Ethernet Boards *do not* have pins for LASER1 and LASER2 at their (10-pin) EXT. 2 socket connectors.

Configuration by Solder Jumpers

The pin (08) of the EXT. 2 socket connector is configured by the solder jumper field C whereas pin (09) is configured by solder jumper field B. Both jumper fields are on the lower side of the RTC6 Ethernet Board, see [Figure 73](#). For further information, see [Section "Solder Jumper Field B – Configuring pin \(09\) of EXT. 2 Socket Connector", page 877](#) and [Section "Solder Jumper Field C – Configuring pin \(08\) of EXT. 2 Socket Connector", page 878](#).

Notes

- If the DATA7 bit (DATA7) is assigned to pin (08), then the full 8-bit output value is available at the output port (pins (1) to pin (08) of the EXT. 2 socket connector).
- If pin (08) is set to +5 V (HIGH level), an offset of 128 results for the output value. That is, the output value range is from 128...255.
- If pin (08) is set to GND (LOW level) the output value range is restricted to 0...127.
- The DATA7 bit can be used for other purposes by assigning it to pin (09).

8-Bit Digital Output Port

As with RTC6 PCIe Board, see [Section "8-Bit Digital Output Port", page 80](#).

16.2.13 Master Socket Connector, Slave Socket Connector

The Master socket connector has 6 pins, see [Figure 87](#). It is located on the upper side of the RTC6 Ethernet Board, see [Figure 72](#), number 10.

The Slave socket connector has 6 pins, see [Figure 87](#). It is located on the lower side of the RTC6 Ethernet Board, see [Figure 72](#), number 16.

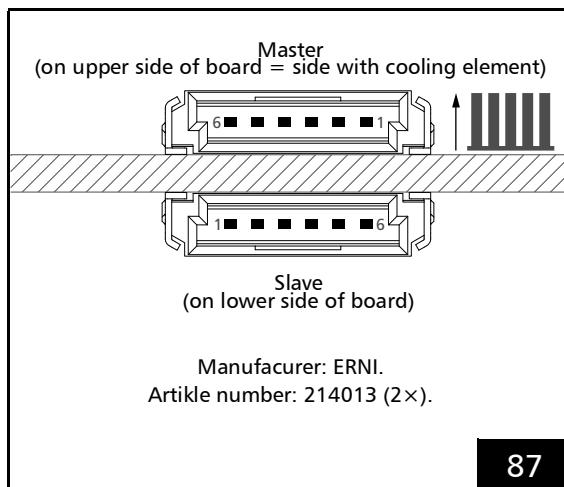
The both socket connectors serve to synchronize the clock cycles of several RTC6 boards.

Connection cables are available from SCANLAB, see [Figure 9, page 65](#). The necessary information for assembling your own cables is shown in [Figure 8, page 65](#).

For further information, see

- [Chapter 6.6.3 "Master/Slave Operation", page 123](#)
- [Chapter 9.3.1 "Starting and Stopping Lists by External Control Signals and Master/Slave Synchronization", page 288](#).

A specific hardware revision of the RTC6 Ethernet Board is required for proper functioning of the master/slave clock synchronization (see label with part number and modification index). If you have an older RTC6 Ethernet Board or are unsure, contact SCANLAB.



Notice!

- Master/Slave-connected RTC6 Ethernet Boards are destroyed by different potentials. This may even be the case, if the boards are connected to the same power supply but the cabling is unfavourable. Refer to [Figure 88](#) for the recommended wiring.

To avoid different potentials:

- (1) Make sure that the power supply is switched off.
- (2) First connect one RTC6 Ethernet Board to the power supply, and then connect the other one through a branch of this cable (keep the length as short as possible).
- (5) Only then switch the power supply back on again.

- Always switch off the power supply of all Master/Slave-connected RTC6 Ethernet Boards before disconnecting the cabling from them.

Master socket connector and Slave socket connector.
The pitch of the pins is 1.27 mm.

16.2.14 Jumper Settings

SCANLAB ships RTC6 Ethernet Boards in various jumper configurations. The jumpers can be reconfigured at a later time. See [Chapter 2.7 "Jumper Settings and Type Designations"](#), page 39.

Notice!

- Only configure allowed jumper settings. Otherwise, the board gets damaged!

Solder Jumper Field A – Configuring the Output Signal Level at EXTENSION 1 Socket Connector

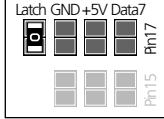
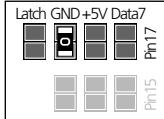
- Position on the board: lower side, see [Figure 73](#), number 12.
- Purpose: to configure the level (5 V or 3.3 V) of all output signals at the EXTENSION 1 socket connector, see the following table.
- See also [Section "Configuring the Output Signal Level", page 77](#).

Allowed jumper setting	At the EXTENSION 1 socket connector
 closed* open	Output signal level 5 V.
 open closed*	Output signal level 3.3 V.
 open open	No signal output.

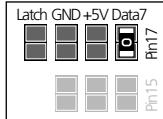
* Caution: make sure that *only one* position is closed in this solder jumper field. Other combinations are not allowed and cause damage to the board!

Solder Jumper Field B – Configuring pin (09) of EXT. 2 Socket Connector

- Position on the board: lower side, see [Figure 73](#), number [13](#).
- Purpose: to configure the signal at pin (09) of the EXT. 2 socket connector, see the following table.
- See also ["Configuration by Solder Jumpers"](#), [page 79](#).
- Configurations of solder jumper field B and solder jumper field C are independent from each other.
- On the RTC6 Ethernet Board the printed label of solder jumper field B is 'Pin 17'. This has been chosen deliberately in order to keep consistency with already existing RTC boards. Even though pin (09) is actually configured.

Allowed jumper setting	Output at the EXT. 2 socket connector pin (09)
 open open open open	No signal.
 closed* open open open	LATCH signal.
 open closed* open open	GROUND (low level).

* Caution: make sure that *only one* position is closed in this solder jumper field. Other combinations are not allowed and cause damage to the board!

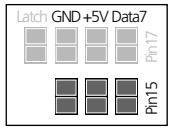
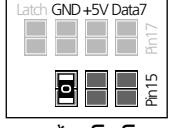
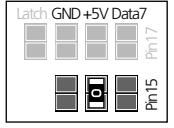
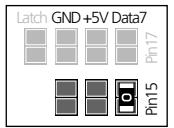
Allowed jumper setting (cont'd)	Output at the EXT. 2 socket connector pin (09) (cont'd)
 open open closed* open	+5 V (high level).
 open open open closed*	DATA7 ^(a) .

* Caution: make sure that *only one* position is closed in this solder jumper field. Other combinations are not allowed and cause damage to the board!

(a) Synonym: Data Bit #7. **MSB** of the 8-bit output value.

Solder Jumper Field C – Configuring pin (08) of EXT. 2 Socket Connector

- Position on the board: lower side, see [Figure 73](#), number **14**.
- Purpose: to configure the signal at pin (08) of the EXT. 2 socket connector, see the following table.
- See also ["Configuration by Solder Jumpers"](#), [page 79](#).
- Configurations of solder jumper field C and solder jumper field B are independent from each other.
- On the RTC6 Ethernet Board the printed label of solder jumper field C is 'Pin 15'. This has been chosen deliberately in order to keep consistency with already existing RTC boards. Even though pin (08) is actually configured.

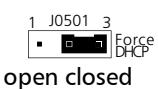
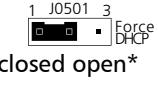
Allowed jumper setting	Output at the EXT. 2 socket connector pin (08)
 open open open	No signal.
 closed* open open	GROUND (low level).
 open closed* open	+5 V (high level).
 open open closed*	DATA7 ^(a) .

*Caution: make sure that *only one* position is closed in this solder jumper field. Other combinations are not allowed and cause damage to the board!

(a) Synonym: Data Bit #7. **MSB** of the 8-bit output value.

Jumper Field 'Force DHCP'

- Position on the board: lower side, see [Figure 72](#), number 11.
- Purpose: see the following table.

Allowed jumper setting	Effect
 open closed	<p>When the RTC6 Ethernet Board is switched on, an IP address is to be obtained by DHCP.^(a) "Force DHCP position"</p> <p>For all configurable network parameters (static IP, pertaining net mask, gateway, UDP and TCP ports), the default settings are used (instead the ones which are saved on the board).</p>
 closed open*	<p>If user-defined network settings are saved, these are used when switching on the RTC6 Ethernet Board.</p>
 open open	<p>Same as "open closed".</p>

* Position as delivered by the factory.

(a) See also [page 881](#).

16.2.15 Real-Time Clock

The RTC6 Ethernet Board features a real-time clock. If the board is without power, the clock continues to run for about > 1 week. With a [Hardware reset](#), this time is automatically adopted (no [time_update](#) required).

By [time_update](#), the real-time clock is automatically synchronized with the PC time. The accuracy can be fine-tuned by [time_control_eth](#).

16.3 Installation and Operation

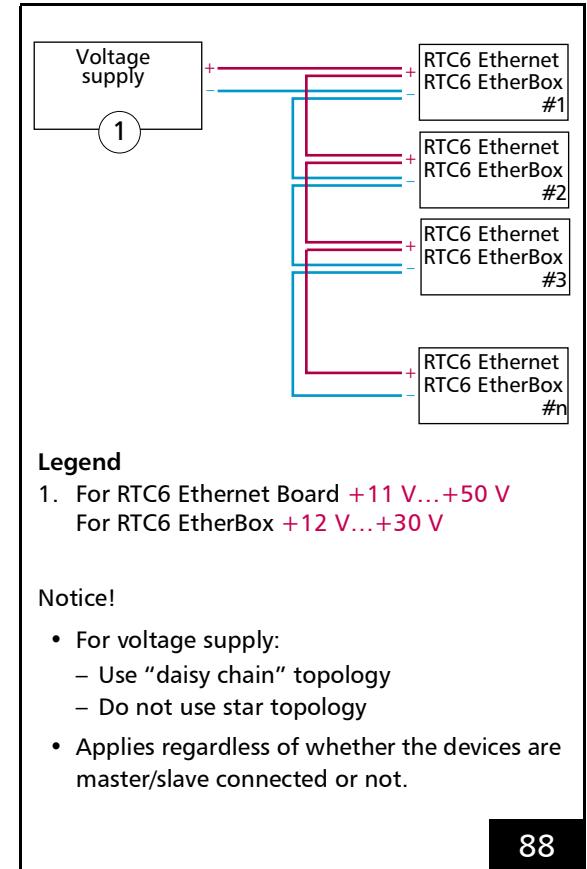
16.3.1 Hardware Installation

Notice!

- Store the board in an electrostatically neutral environment using the supplied anti-static bag.
- Observe ESD precautions when installing the board.
- Do not touch the electrical contacts of the board.
- Protect the board from moisture, dust, corrosive vapors and mechanical loads.

Proceed as follows to install the RTC6 Ethernet Board in a shielded housing:

- (1) Remove the RTC6 Ethernet Board from its anti-static bag. Do not touch the electrical contacts of the board.
- (2) Mount the RTC6 Ethernet Board at the intended location in your shielded housing.
- (3) Connect the RTC6 Ethernet Board to:
 - your power supply using a power cable
 - Also observe [Figure 88](#) –
 - your Ethernet port using an Ethernet cable
- (4) Connect the RTC6 Ethernet Board using appropriate cables to:
 - the scan head
 - the laser
- (5) If you want to use the signals at the RTC6 Ethernet Board socket connectors, then attach appropriate cables.



88

Recommended cabling for connecting several RTC6 Ethernet Boards/RTC6 EtherBoxes.



16.3.2 Software Installation

For installing and starting-up an RTC6 Ethernet Board and its software proceed analogously as with RTC6 PCIe Boards, [Chapter 5 "Installation and Start-Up", page 87](#). However, step 3, [page 87](#) does not apply:

RTC6 Ethernet Boards do not require the RTC6 board driver.

16.3.3 Connecting to a Network

Connect the RTC6 Ethernet Board with the desired network segment and switch on the supply voltage. The green LED at the ETH connector, see [Chapter 16.2.7 "ETH Connector", page 869](#), should now light up and possibly blink.

If no static IP address is stored on the RTC6 Ethernet Board, a DHCP server is necessary to obtain a dynamic IP address. In this case the yellow LED should light up after a few seconds. This indicates a successful receipt of an IP address.

If in "Force DHCP position" (as of [BIOS 24](#)) no IP address has been assigned within 60 seconds after Power-On, then the link-local address 169.254.1.0/16 is automatically used.

If a static IP address is stored on the RTC6 Ethernet Board, the yellow LED immediately lights up after switching on the RTC6 Ethernet Board.

Important: it is possible that the IP configuration stored on the RTC6 Ethernet Board is invalid for the connected network segment. In this case the yellow LED lights up as well. However, the RTC6 Ethernet Board is unresponsive in the network.



16.4 Notes on Migrating Existing and Programming New RTC6 User Programs

RTC6 Ethernet Boards must be registered manually to the RTC6 board management, see [Chapter 16.5.3 "About the RTC6 Board Management", page 887](#) (`init_rtc6_dll` only searches RTC6 PCIe Boards).

After that, RTC6 Ethernet Boards can be used generally in the same way as RTC6 PCIe Boards.

In principle, communication via Ethernet is more unsteady compared to a PCIe bus connection within a PC. Therefore, further attention should be paid to potentially occurring errors and to its handling than with RTC6 PCIe Boards (see `get_last_error`).

16.4.1 Finding RTC6 Ethernet Boards in the Network and Querying their Properties

Case 1: IP address of the RTC6 Ethernet Board is known

```
card_no = eth_assign_card_ip(ip, 0); // 0: assign to first free number
Result = select_rtc(card_no);           // If Result equals CardNo: Success
...

```

Case 2: IP address of the RTC6 Ethernet Board is not known

```
result = eth_search_cards(eth_convert_string_to_ip("192.168.250.1"),
                         eth_convert_string_to_ip("255.255.255.0"));
if (result != 0)
{
    card_no = eth_assign_card(1, 0); // take first found card, assign to first free number
    result = select_rtc(card_no);
    if (result == card_no)
    {
        ip = eth_get_ip(); // now IP address is known
        ... // do anything
    }
}
```



16.4.2 Example Code (C++): Initialization Covering RTC6 Ethernet Boards

```
// Abstract
//      A console application to demonstrate how to initialize RTC6 boards
//
// Comment
//      This application demonstrates how to properly initialize RTC6 boards.
//      It will enumerate all available PCIe boards as well as search for Ethernet boards in a given subnet.
//
// Platform
//      Win32 - 32-bit Windows
//      x64   - 64-bit Windows
//
// Necessary Files
//      RTC6impl.h
//      RTC6DLL.dll for Win32 or RTC6DLLx64.dll for x64
//      RTC6DLL.lib for Win32 or RTC6DLLx64.lib for x64
//      RTC6DAT.dat
//      RTC6OUT.out for PCIe boards and/or RTC6ETH.out for Ethernet boards
//      RTC6RBF.rbf
//      Cor_1to1.ct5 or any other correction file
//
// Compiler
//      - tested with Microsoft C++ Compiler 19.00.24215.1.

#include "RTC6impl.hpp"

#include <iostream>
#include <array>
#include <string>
#include <conio.h>

using namespace std;

// RTC error codes
const UINT ERROR_NO_ERROR = 0U;
const UINT ERROR_NO_CARD = 1U;
const UINT ERROR_VERSION_MISMATCH = 256U;

// Use current working directory as path
const char* PROGRAM_FILE_PATH = nullptr;
// Use Cor_1to1.ct5 in current working directory as correction file
const char* CORRECTION_FILE_PATH = "./Cor_1to1.ct5";

// Subnet to use for Ethernet board search
const char* ETH_SEARCH_IP = "192.168.250.253";
const char* ETH_SEARCH_NETMASK = "255.255.255.0";

bool LoadProgramAndCorrectionFile(UINT card)
{
    // DAT, ETH/OUT and RBF need to be in the current working directory
    const auto loadProgram = n_load_program_file(card, PROGRAM_FILE_PATH);
    if (loadProgram != ERROR_NO_ERROR)
    {
        cout << "n load_program_file for card " << card << " failed with error code: " << loadProgram << endl;
        return false;
    }

    // Acquire board for further access
    const auto acquire = acquire_rtc(card);
    if (acquire != card)
    {
        cout << "acquire_rtc for card " << card << " failed with error code: " << acquire << endl;
        return false;
    }
}
```



```
// Load 2D correction file as table 1
const auto loadCorrection = n_load_correction_file(card, CORRECTION_FILE_PATH, 1, 2);
if (loadCorrection != ERROR_NO_ERROR)
{
    cout << "n_load_correction_file for card " << card << " failed with error code: " << loadCorrection << endl;
    return false;
}

// select_cor_table( 1, 0 ); is done internally. Call select_cor_table, if you want a different setting.

const auto serialNumber = n_get_serial_number(card);
cout << "Initialized card " << card << " (SN " << serialNumber << ")" << endl;

return true;
}

bool InitializeRTC()
{
    // Initialize DLL
    const auto initDLL = init_rtc6_dll();
    if (initDLL != ERROR_NO_ERROR)
    {
        if (initDLL & ERROR_VERSION_MISMATCH)
        {
            // Version mismatch can happen if a board has been previously used with a different software version.
            // This error can be fixed by loading the current program file on to the board.
        }
        else if (initDLL & ERROR_NO_CARD)
        {
            // The RTC6 DLL will return ERROR_NO_CARD if no PCIe board has been found.
            // We can still use Ethernet boards if available.
        }
        else
        {
            cout << "init_rtc6_dll failed with error code: " << initDLL << endl;
            return false;
        }
    }

    // Initialize PCIe boards
    const auto foundCards = rtc6_count_cards();
    for (auto card = 1; card <= foundCards; card++)
    {
        if (!LoadProgramAndCorrectionFile(card))
        {
            return false;
        }
    }
}
```



```
// Search for and initialize Ethernet boards
const auto ethSearchIP = eth_convert_string_to_ip(ETH_SEARCH_IP);
const auto ethSearchNetmask = eth_convert_string_to_ip(ETH_SEARCH_NETMASK);
const auto foundEthCards = eth_search_cards(ethSearchIP, ethSearchNetmask);
if (foundEthCards > 0)
{
    for (auto searchCard = 1; searchCard <= foundEthCards; searchCard++)
    {
        array<UINT, 16> cardInfo;
        eth_get_card_info_search(searchCard, (UINT)cardInfo.data());
        const auto serialNumber = cardInfo[1];
        // Param 0 automatically assigns the board to the next free index
        const auto card = eth_assign_card(searchCard, 0);
        if (card <= 0)
        {
            cout << "eth_assign_card for SN " << serialNumber << " failed with error code: " << card;
            return false;
        }
        if (!LoadProgramAndCorrectionFile(card))
        {
            return false;
        }
    }
}
// At least one PCIe or Ethernet board available
return (foundCards > 0) || (foundEthCards > 0);
}

int main(int argc, char* argv[])
{
    if (!InitializeRTC())
    {
        return 1;
    }
    // Use boards for marking etc...
    getch();
    return 0;
}
```

16.5 RTC6 Ethernet Board Commands and Functions

16.5.1 Notes on Working with IP Addresses

With the RTC6 commands, all IP addresses (always IPv4, IPv6 is not supported) are specified as decimal values in Big Endian format ("Big Endian byte order").

For example, the IP address "192.168.250.1" must be specified as "33204416", see following table.

Format	IP address	Hex value	Decimal value
Little Endian	192.168.250.1 (a)	0xC0A8FA01	3232299521
Big Endian	1.250.168.192	0x01FAA8C0	33204416

(a) Usual dotted decimal notation.

eth_convert_ip_to_string converts the IP address in Big Endian byte order to usual dotted decimal notation.

eth_convert_string_to_ip converts the IP address in usual dotted decimal notation to Big Endian byte order.

16.5.2 About Searching RTC6 Ethernet Boards

To search for all RTC6 Ethernet Boards available in the network, **eth_search_cards** can be used. If the search is to be limited to a certain address range only, then **eth_search_cards_range** is to be used.

The data of all RTC6 Ethernet Boards which have been answered within a configurable timeout (see **eth_set_search_cards_timeout**) are registered to the search result list (see below).

By a card search, RTC6 Ethernet Boards with unknown IP address can be identified in the network (for example, because they have received it dynamically by a DHCP server).

The found RTC6 Ethernet Boards are registered in a temporary list which is the search result list.

Index	Record in the search result list
1	Information ^(a) on RTC6 Ethernet Board 1
2	Information ^(a) on RTC6 Ethernet Board 2
n	Information ^(a) on RTC6 Ethernet Board n

(a) IP address, serial number, connection status, etc., see **eth_get_card_info_search**.

For a specified search result list index **eth_get_card_info_search** returns the available information on the RTC6 Ethernet Board, whereas **eth_get_ip_search** returns only the IP address and **eth_get_serial_search** only the serial number.

On the one hand the number of found RTC6 Ethernet Boards is already returned by **eth_search_cards**. On the other hand **eth_found_cards** also returns it at any time as well (without the need to perform the search in the network again).

Several subsequent calls of **eth_search_cards** can deliver different results, depending how many RTC6 Ethernet Boards are available in the network and in which chronological order the answers come in.

16.5.3 About the RTC6 Board Management

RTC6 boards are addressed by a unique number under which they must be entered in the RTC6 board management.

The RTC6 board management is an [RTC6 DLL](#)-internal list consisting of 255 possible RTC6 board records, see following table.

Index	RTC6 board record
1	RTC6 PCIe Board 1
2	RTC6 PCIe Board 2
3	RTC6 PCIe Board 3
4	"No card"
5	Information on RTC6 Ethernet Board ^(a)
6	Information on RTC6 Ethernet Board ^(a)
7	"No card"
...	...
42	Information on RTC6 Ethernet Board ^(a)
43	"No card"
...	...
255	"No card"

(a) IIP address, serial number, connection status, etc., see [eth_get_card_info_search](#).

At the beginning of the list are RTC6 PCIe Boards, if any are present in the PC.

RTC6 PCIe Boards are automatically searched by [init_rtc6_dll](#) and consecutively numbered. The numbering cannot be changed. [rtc6_count_cards](#) only returns the number of RTC6 PCIe Boards found.

RTC6 Ethernet Board, on the other hand, must be entered manually into the RTC6 board management:

- If the IP address is known, [eth_assign_card_ip](#) can be used to enter a card at any index between [rtc6_count_cards](#) + 1...255.
- If the IP address is unknown, by [eth_assign_card](#) an RTC6 Ethernet Board from the search result list, see [Chapter 16.5.2 "About Searching RTC6 Ethernet Boards", page 886](#), can be entered at an arbitrary index. However, neither an RTC6 PCIe Board nor an RTC6 Ethernet Board must be registered at this index

[eth_max_card](#) returns the highest index where an RTC6 Ethernet Board in the RTC6 board management is registered.

In contrast, [eth_count_cards](#) returns the total number of entered RTC6 Ethernet Boards.

By [get_card_type](#) the registered board type can be queried:

- 0 = "No card"
- 1 = RTC6 PCIe Board
- 2 = RTC6 Ethernet Board



16.5.4 Checking the Connection to the RTC6 Ethernet Board

In general, communication via Ethernet is more unreliable than PCIe bus connections within PCs (simple example: Ethernet cable is not plugged in). By `eth_check_connection` it can be simply checked, whether the RTC6 Ethernet Board responds and therefore, the Ethernet connection still exists.

The behavior of the RTC6 Ethernet Board in case of `Ethernet Link Loss` can be set with `eth_configure_link_loss`. In `Mode` = 0...2, among other things, Bit #15 of `get_startstop_info` is set with an `Ethernet Link Loss` (detection time: < 100 ms). Bit #15 is set back to 0 not until `get_startstop_info` is called.

16.5.5 Command Set for the RTC6 Ethernet Board

- See [Chapter "Control Commands for RTC6 Ethernet Boards", page 308](#).

16.6 Safe Startup and Shutdown Sequences

To assure safety during startup, switch on the components of the laser system in the following order:

- (1) Switch on the network PC.
- (2) Switch on the power supply for the RTC6 Ethernet Board.
- (3) Start the control software.
- (4) Switch on any required peripheral devices.
- (5) Switch on the power supply for the scan system.
- (6) Switch on the laser.

To assure safety during shutdown, switch off the components of the laser system in exactly the reverse order:

- (1) Switch off the laser.
- (2) Switch off the power supply for the scan system.
- (3) Switch off the peripheral devices.
- (4) Terminate the control software.
- (5) Switch off the power supply for the RTC6 Ethernet Board.
- (6) Shut down the network PC.



Caution!

- When the PC switches on or off, the RTC6 Ethernet Board board output ports levels might briefly fluctuate, resulting in unintended changes to laser control signals. The above-mentioned startup and shutdown sequences must therefore be strictly followed. Otherwise, the laser might briefly, unexpectedly and dangerously switch on.
- Always start up the PC and control software prior to turning on the scan system. And switch the scan system back off prior to shutting down the control software and PC. Otherwise, unintended scan system motions might occur. The laser must always be switched on last and switched off first. Otherwise, there is the risk that the laser beam might be deflected in an arbitrary direction.

16.7 Standalone Functionality

- In contrast to PC operation, the aim of the **Standalone Operation Mode** is that the RTC6 Ethernet Board operates independently without a connected PC. Among other things, for this purpose, the list commands to be processed, all correction tables to be used and the control commands to be executed during automatic booting⁽¹⁾ must be stored in the **NAND memory**.

Notice!

- The following requirements must be met for **Standalone Operation Mode**:
 - (1) At least **BIOS** version 0x26 (**BIOS-ETH 26**) is installed^(a) on the RTC6 Ethernet Board. See [Chapter 16.7.1 "Upgrading BIOS-ETH", page 891](#).
 - (2) The user program uses RTC6 Software Package \geq **1.7.0**, that is, a combination of
 - \geq **DLL 618**
 - \geq **ETH 618**
 - \geq **RFB 623**
 - \geq **DAT 603**

(a) See [RTC6conf.exe](#) and [RTC6BIOSETH_35.out](#).

- Then the RTC6 Ethernet Board can be configured to boot automatically⁽¹⁾ (depending on the data stored in **NAND memory** by **store_program(Mode)** after a **Hardware reset**. After that, it is in one of the following states:
 - "Normal PC Operation State"
 - "Standalone Basic State"
 - "Standalone Full State"
- The **"Normal PC Operation State"** is achieved by:
 - **BIOS** $<$ **BIOS-ETH 26**
 - \geq **BIOS-ETH 26** and **set_eth_boot_control(0)**
 - \geq **BIOS-ETH 26** and **store_program(1)**
 In these cases, **load_program_file** must be called for further operation.

(1) In this Chapter, "automatic booting" means that in addition to the actual booting, additional data is read out from the **NAND memory**.

- The **"Standalone Basic State"** is achieved by:
 - **BIOS** \geq **BIOS-ETH 26**, **store_program(0)** and **set_eth_boot_control(1)**
 A **load_program_file** call is not required. See also [Chapter 16.7.4 "Boot Image", page 893](#). The procedure for **"Standalone Basic State"** is described in [Chapter 16.7.2 "Preparing the "Standalone Basic State""](#), page 891.
- The **"Standalone Full State"** is achieved by:
 - **BIOS** \geq **BIOS-ETH 26**, **store_program(2)** and **set_eth_boot_control(1)**
 Compared to the **"Standalone Basic State"** it is also no longer necessary to load correction files. The same applies to loading of the list commands which have been stored upon **store_program(2)**. Furthermore, control commands for configuration⁽²⁾ can be executed automatically as required. See also [Chapter 16.7.4 "Boot Image", page 893](#). The procedure for **"Standalone Full State"** is described in [Chapter 16.7.3 "Preparing the "Standalone Full State""](#), page 892. In **"Standalone Full State"**, the RTC6 Ethernet Board can process its list independently and without a connected PC after an /START.
- The **NAND memory** contents can be written to the PC as a so-called **"Boot Image"** by **read_image_eth** as a binary file and copied to any number of RTC6 Ethernet Boards by **write_image_eth**, see [Chapter 16.7.4 "Boot Image", page 893](#).
- Details on automatic booting can be found in [Chapter 16.7.6 "Automatic Booting – Process in Detail", page 896](#).
- RTC4 SCANalone Board users must observe the safety notice on the encoder counter direction, see [Notice!, page 56](#).

(2) See [Chapter 16.7.5 "Control Commands Allowed for Automatic Booting", page 894](#).

16.7.1 Upgrading BIOS-ETH

Proceed as follows to upgrade the **BIOS** of an RTC6 Ethernet Board with \leq **BIOS-ETH 26** to **BIOS-ETH 35** only (no subsequent **Standalone Operation Mode**):

- (1) Carry out a **Hardware reset**.
- (2) Call **load_program_file** by specifying the relevant data from RTC6 Software Package \geq V1.7.0.
- (3) Call **store_program(Mode = 1)**.
The **NAND memory** content is erased.
- (4) Call **set_eth_boot_control(0)**.
Thus, the board does *not* boot automatically⁽¹⁾ after a **Hardware reset** (no matter what is stored in **NAND memory**).
- (5) Run **RTC6conf.exe** and install **RTC6BIOSETH_35.out** via **Upgrade BIOS** button.
Thus, the upgrade is completed.
- (6) Carry out a **Hardware reset**.
 - The RTC6 Ethernet Board *does not* boot automatically⁽¹⁾
 - Is now ready for PC operation

16.7.2 Preparing the "Standalone Basic State"

Proceed as follows to put the board to **"Standalone Basic State"**:

- (1) Prerequisite: Chapter 16.7.1 "Upgrading BIOS-ETH", page 891 has been carried out.
- (2) Call **load_program_file** by specifying the relevant data from RTC6 Software Package \geq V1.7.0.
- (3) Call **store_program(Mode = 0)**.
Data for the **"Standalone Basic State"**, see page 782, is saved to the **NAND memory**.
- (4) Call **set_eth_boot_control(Ctrl = 1)**.
Thus, the board boots automatically after a **Hardware reset**⁽¹⁾⁽²⁾.
- (5) Carry out a **Hardware reset**.
 - The RTC6 Ethernet Board boots automatically⁽¹⁾
 - Is now ready for PC operation
 - Subsequently, you do *not* need to call **load_program_file**

Notes

- The actual **RTC6 files** no longer need to be provided in the user program. An accidentally wrong **RTC6 DLL** version is automatically detected, because a wrong **RTC6 DLL** leads to a **get_last_error** return code **RTC6_VERSION_MISMATCH** (whereas a **load_program_file** does not, as long as **RTC6 files** are compatible with the **RTC6 DLL**).

(1) See Footnote, page 890.

(2) This is also the default setting with "new" (= **set_eth_boot_control(Ctrl = 0)** has never been executed) RTC6 Ethernet Boards.

16.7.3 Preparing the "Standalone Full State"

Proceed as follows to put the board to "Standalone Full State":

- (1) Prerequisite: Chapter 16.7.2 "Preparing the "Standalone Basic State""", page 891 has been carried out.
- (2) Verify that the RTC6 Ethernet Board works properly in PC operation.
- (3) Work out a suitable sequence of control commands for automatic booting⁽¹⁾.
 - The allowed control commands are listed in Chapter 16.7.5 "Control Commands Allowed for Automatic Booting", page 894, separated according to **Boot Phase 1** and **Boot Phase 2**.
 - Call **eth_boot_dcmd** before each of these commands.
 - Pay attention to the correct order, for example,
 - **config_list** should be called before loading list commands
 - if **set_scanahead_params** is used, a scan system of the excelliSCAN series must already be connected and switched on
 - if an "Automatic Laser Control" is used, it may only be activated after **set_scanahead_params**
- (4) Test the result of step 3:
execute it in PC operation in exactly the same way as it is to be executed later in **Standalone Operation Mode**.
- (5) Call **load_program_file** in order to initialize the RTC6 Ethernet Board.
- (6) Load all required control commands, list commands and correction files.
- (7) Emergency provision in case an error occurs during saving: call **set_eth_boot_control(0)**. This prevents automatic booting.
- (8) Call **store_program(Mode = 2)**. The process can take up to 2 minutes.
- (9) If an error occurred:
Call **store_program(Mode = 2)** again.
- (10) If the process has been completed without errors: Call **set_eth_boot_control(Ctrl = 1)**. This activates automatic booting.
- (11) Carry out a **Hardware reset**.
 - The RTC6 Ethernet Board is now in "Standalone Full State"
 - See also Chapter 16.7.6 "Automatic Booting – Process in Detail", page 896

Notes

- In "Standalone Full State" the RTC6 Ethernet Board is already configured with the control commands for automatic booting⁽²⁾ and provided with all correction tables. If list commands have been stored, they are ready to be executed automatically after an /START.

(1) See Footnote, page 890.

(2) See Chapter 16.7.5 "Control Commands Allowed for Automatic Booting", page 894.

16.7.4 Boot Image

- Prerequisites: see [page 890](#).

The data in the **NAND memory** can be written to the PC as “boot image” and copied from there to any RTC6 Ethernet Boards (for example, of several production machines):

- **read_image_eth**,
see [Creating a Boot Image on the PC](#)
- **write_image_eth**,
see [Copying Boot Image to Board\(s\)](#)

The properties of boot images are shown in Table 7 (RTC6 Software Package V1.7.0, approximate values):

Table 7: Properties of Boot Images

Boot Image	File size	Reading duration ^(a)	Writing duration ^(b)
for “Standalone Basic State”	2 MB	a few seconds	a few seconds
for “Standalone Full State”	160 MB ... 264 MB	up to 2 minutes	up to 2 minutes

(a) With **read_image_eth**.

(b) With **write_image_eth**.

Creating a Boot Image on the PC

- Prerequisites: see [page 890](#).

read_image_eth(*Name*) read out the contents of the **NAND memory** and writes it in binary to the “*Name*” file on the PC. The user program must have write permission for this file. For reading duration, see Table 7.

See also [Section “Procedure after an Transmission Abortion”, page 893](#).

Copying Boot Image to Board(s)

- Prerequisites: see [page 890](#).

write_image_eth(*Name*) reads the file “*Name*” and writes its content to the **NAND memory**. The user program must have read permission for this file. For writing duration, see Table 7.

- (1) Call **set_eth_boot_control(0)**.
Thus, the board *does not* boot automatically after a **Hardware reset** (recommended as automatic booting might be faulty).
- (2) Call **write_image_eth(*Name*)** and do not interrupt the process! Otherwise, observe [Section “Procedure after an Transmission Abortion”, page 893](#).
- (3) If the process went without errors, call **set_eth_boot_control(1)**. Otherwise, repeat step 2.
- (4) Carry out a **Hardware reset**.
Provided the same options are enabled: This RTC6 Ethernet Board boots exactly the same way as the RTC6 Ethernet Board from which the boot image has been taken.

Procedure after an Transmission Abortion

If the transmission is aborted, for example, due to an Ethernet connection interruption, the RTC6 Ethernet Board probably remains in an **INTERNAL-BUSY list execution status** state. The aborted process is not continued, even if the connection is re-established:

- (1) Call either **stop_execution** or execute an /STOP in order to release the board from this **INTERNAL-BUSY list execution status** state.
- (2) Call **read_image_eth(*Name*)** (once again).
Thus, the board *does not* boot automatically after a **Hardware reset**.
- (3) Carry out a **Hardware reset**.
- (4) Call **load_program_file**.
- (5) If you
 - create a boot image on the PC:
call **read_image_eth(*Name*)** again.
 - copy a boot image to the board:
call **write_image_eth(*Name*)**.

16.7.5 Control Commands Allowed for Automatic Booting

- Only certain control commands are allowed in each boot phase, see table 8 and [Chapter 16.7.6 "Automatic Booting – Process in Detail", page 896](#):
 - [Boot Phase 1](#)
 - [Boot Phase 2](#)
- They meet one or more of the following criteria:
 - Requires no communication with the PC
 - Does not send a response to the PC
 - There is no corresponding list command
- Exceptions are, for example,
 - [set_free_variable](#), see [Comments](#) there
 - [set_auto_laser_control](#) or [set_scanahead_params](#)
 - They check the connected scan systems at point in time of the call
 - They save exclusively the parameters to be finally set only in case of success
 - [set_jump_mode](#) only sets the tuning numbers [VA1...JB2](#) during automatic booting and does not check whether they match the scan system.
 - The table containing the [Jump Delay](#) values [\[JumpTable<No>\]](#), see [page 218](#) and [load_jump_table_offset](#), must have been previously saved via [create_dat_file](#).
- Upon (later) automatic booting, the connected scan systems are *not* checked anymore!
- [Boot Phase 2](#)-control commands are control commands which:
 - Necessarily need to be executed after a [Boot Phase 1](#)-control command
 - Require functioning peripherals, for example, an [iDRIVE](#) scan system⁽¹⁾ with "Automatic Laser Control". Important: users must make sure themselves that the periphery is actually working at runtime

(1) See Glossary entry on [page 26](#).



Table 8: Control commands allowed for automatic booting (alphabetically)

Boot Phase 1

bounce_supp
config_laser_signals
config_list
eth_boot_timeout
home_position
home_position_xyz
mcbsp_init
set_control_mode
set_extstartpos
set_free_variable
set_jump_mode
set_laser_control
set_laser_mode
set_matrix(HeadNo = 4)
set_max_counts
set_mcbsp_freq
set_mcbsp_out_ptr
set_offset_xyz(HeadNo = 4)
set_pulse_picking_length
set_rot_center
set_scanahead_params
set_scanahead_speed_control
set_timelag_compensation
simulate_encoder
uart_config

Boot Phase 2

control_command
init_fly_2d
select_cor_table
set_auto_laser_control
set_dsp_mode
set_hi
start_loop

16.7.6 Automatic Booting – Process in Detail

(1) After a **Hardware reset**, the RTC6 Ethernet Board tries to read data for automatic booting from the **NAND memory**.⁽¹⁾

During this time the yellow LED flashes dimmed with approx. 1 Hz until the end of the initialization.

- Case 1: There are no data present (after **store_program(Mode= 1)** or **set_eth_boot_control(0)** has been carried out.

The initialization is completed without these data. Afterwards, the RTC6 Ethernet Board is in **"Normal PC Operation State"**, page 890.

- Case 2: There are only data for **"Standalone Basic State"**, see [page 782](#) (after **store_program(Mode= 0)**). The RTC6 Ethernet Board is initialized (as by **load_program_file**). Afterwards, it is immediately ready for PC operation = **"Standalone Basic State"**.
- Case 3: There are also data for **"Standalone Full State"**, see [page 782](#) (after **store_program(Mode= 2)**). Then step 2 is executed.

(2) The RTC6 Ethernet Board

- switches on the **BUSY pin**
- reads the “remaining” data for automatic booting to **"Standalone Full State"**, see [page 782](#) from the **NAND memory**.⁽¹⁾
- The subsequent boot steps 3...6 are two-phase.

(3) In **Boot Phase 1** (duration: some seconds):

- The correction file(s) are read out
- **Boot Phase 1**-control commands are executed (in the order in which they have been saved). If such commands are stored more than once, they are executed correspondingly often, with newer data overwriting older data.

(4) After **Boot Phase 1**:⁽¹⁾

- Initialization ceases
- The **BUSY pin** is switched off
- Do not switch the laser on yet!
- Switch on scan systems and other peripheral devices
- Make sure peripheral devices are ready for operation. Only then, trigger **Boot Phase 2** by an **/START**
- If your peripheral devices are always ready for operation after a certain time, you can specify a waiting time (**TimeOut**) with the **Boot Phase 1**-control command **eth_boot_timeout**. After its expiry, initialization continues automatically (= without an explicit **/START**). With an **/START** within the waiting time, **Boot Phase 2** starts immediately. Outside the waiting time, an **/START** is ignored as long as **Boot Phase 2** is still running (see **BUSY pin!**) The **/START** is automatically enabled (similar to **set_control_mode(Bit #0 = 1)**). Further **/STARTs** after an **/STOP** must be enabled by users themselves by **set_control_mode(Bit #3 = 1)**.

(1) **Status == 1** of **eth_get_standalone_status**

(5) In **Boot Phase 2**:⁽¹⁾

- The **BUSY pin** is switched on
- The **RTC6 List Memory** is read out (duration: approx. 1 minute)
- **Boot Phase 2-control commands** are executed.
- Initialization ceases
- The **BUSY pin** is switched off

(6) After **Boot Phase 2**:

- If an error has occurred⁽²⁾ (further operation of the board may not be possible), the LED continues to flash at about 4 Hz
- When the initialization is successfully completed⁽³⁾, the dimmed LED flashing stops. The LED returns to its normal state, see [Chapter 16.2.7 "ETH Connector", page 869](#)
- The target state of the RTC6 Ethernet Board indicates `eth_get_standalone_status(&Mode =1...2)`.
- At this point, it would be a suitable point in time to also switch on the laser (observe the notices on laser safety in [Chapter 3.2 "Laser Safety", page 62](#)) as well as other peripherals.
- The RTC6 Ethernet Board waits endlessly for an /START, which starts the processing of the list
- The RTC6 Ethernet Board is ready for PC operation as well

16.8 "High Performance Mode"

With RTC6 Ethernet Boards, `eth_set_high_performance_mode` switches on (off) the

- "High Performance Mode"

"High Performance Mode" is only effective with list commands. If it is switched on (`Mode = 1`), the **RTC6 DLL** no longer waits for a response from the RTC6 Ethernet Board after each telegram.

Instead, the **RTC6 DLL** now acknowledges several telegrams at a time. Therefore, more list commands per time can be written into the list memory of the RTC6 Ethernet Board (= "higher list download performance").

With control command calls, the **RTC6 DLL** always waits until the RTC6 Ethernet Board has acknowledged all previous list commands.

However, a significantly higher rate can only be achieved, if you:

- Write correspondingly larger list command blocks
- Avoid that by calling a control command the list command blocks are confirmed prematurely

(1) `eth_get_standalone_status(&Status == 1)`

(2) `eth_get_standalone_status(&Error = 1)`

(3) `eth_get_standalone_status(&Status =0)`



16.9 Legal

16.9.1 EU Declaration of Conformity – RTC6 Ethernet Board



EU-Konformitätserklärung

für das Produkt:

RTC4 Ethernet, RTC6 Ethernet

Der Hersteller

SCANLAB GmbH, Siemensstr. 2a, 82178 Puchheim, Deutschland

erklärt, dass das genannte Produkt die einschlägigen Harmonisierungsrechtsvorschriften der Union erfüllt:

- **2011/65/EU** - Richtlinie des EUROPÄISCHEN PARLAMENTS UND DES RATES zur Beschränkung der Verwendung bestimmter gefährlicher Stoffe in Elektro- und Elektronikgeräten (RoHS-Richtlinie).

Folgende harmonisierte Normen wurden angewandt:

- **DIN EN IEC 63000:2019-05** - Technische Dokumentation zur Beurteilung von Elektro- und Elektronikgeräten hinsichtlich der Beschränkung gefährlicher Stoffe

Die alleinige Verantwortung für die Ausstellung dieser Konformitätserklärung trägt der Hersteller.

Dieses Dokument zur Kundeninformation wurde elektronisch ausgestellt und ist daher ohne Unterschrift gültig. Eine unterschriebene Ausführung ist Bestandteil der SCANLAB-internen technischen Dokumentation.

EU Declaration of Conformity

for the product:

RTC4 Ethernet, RTC6 Ethernet

The manufacturer

SCANLAB GmbH, Siemensstr. 2a, 82178 Puchheim, Germany

declares that the product mentioned above complies with the relevant Union harmonization legislation:

- **2011/65/EU** - Directive of the EUROPEAN PARLIAMENT AND OF THE COUNCIL on the restriction of the use of certain hazardous substances in electrical and electronic equipment (RoHS Directive).

The following harmonized standards have been applied:

- **DIN EN IEC 63000:2019-05** - Technical documentation for the assessment of electrical and electronic products with respect to the restriction of hazardous substances

This declaration of conformity is issued under the sole responsibility of the manufacturer.

This document, used for customer information, was issued electronically and is therefore valid without a signature. A signed version is part of the SCANLAB internal technical documentation.

SCANLAB GmbH
Puchheim

2021-07-19

© SCANLAB GmbH – 2021/07



16.9.2 Compliance with FCC Rules

The RTC6 Ethernet 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 RTC6 Ethernet Board is operated in a commercial environment. The RTC6 Ethernet 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 RTC6 Ethernet 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.

16.9.3 TI-RTOS

TI-RTOS is used in the RTC6 Ethernet Board.

TI-RTOS is licenced under the BSD licence:
Copyright (c) 2000-2018, Texas Instruments Incorporated
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of Texas Instruments Incorporated nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

16.10 Technical Specifications – RTC6 Ethernet Board

System Requirements

Windows PC with Ethernet interface
(Gigabit Ethernet (preferred), 10/100 Mbit/s
Ethernet)

Operating system Microsoft Windows 10, 8,
7 as 32 bit or 64 bit
version.

Dimensions (without plugged-in plug connectors)

Length	120 mm
Width	103 mm
Height	Approx. 30 mm

Other Connections and Specifications

Voltage supply	+11 V...+50 V
Maximum power consumption, no peripherals attached	< 10 W

In order to minimize power loss, a low input voltage should be selected.

Operating temperature

- with natural convection 10 °C...50 °C
- with forced convection 10 °C...60 °C

The user must ensure sufficient cooling of the RTC6 Ethernet Board.

Standalone functionality	Yes, see Chapter 16.7 "Standalone Functionality ", page 890.
-----------------------------	--

Interface to the Network

Ethernet	Gigabit Ethernet (preferred), 10/100 Mbit/s Ethernet
IP address obtaining	<ul style="list-style-type: none"> • By a DHCP server • Link-local address (as of BIOS 24)^(a) • Static IP address (storable on the board)
Required local ports ^(b)	<ul style="list-style-type: none"> • 63749 (UDP) • 63750 (UDP, TCP)

(a) See [page 881](#).

(b) Configurable by [eth_set_port_numbers](#).

Scan System Control

Number of list memory Up to 3, configurable
areas

Total capacity of the list 8,388,608 list positions
buffer

Output interval of 10 µs
[Microsteps](#)

Maximum value range -524,288...+524,287
for the [Image field](#) (20 bit, signed)
coordinates

Virtual [Image field](#), e.g. -268,435,456...
for +268,435,455
Processing-on-the-fly (28 bit, signed
= 29 bit)^(a)

(a) With RTC6 Software Package ≥ 1.4.0.

Interfaces to Scan Systems

SCANHEADs Socket Connector

Connector	20 pin socket connector with 2.54 mm pitch of the pins ^(a) . The signals for the first scan head are outputted at pin (01)...pin (10). The signals for the second scan head are outputted at pin (11)...pin (20):
Signal transmission	SL2-100 protocol. Via XY2-100 Converter (Accessory): XY2-100 protocol

(a) Not as with RTC6 PCIe Board.

LASER Socket Connector

Connector	16 pin socket connector with 2.54 mm pitch of the pins ^(a)
laser control signals	LASER1, LASER2, LASERON
• TTL level	5 V, active-HIGH or active-LOW programmable
• Max. current load	20 mA
• Reference	GND - boards without Option "DC/DC Converter". GND2 - boards with Option "DC/DC Converter"
Analog output ports	ANALOG OUT1, ANALOG OUT2 ^(b)
• Output voltage range	0 V...10 V
• Resolution	12 Bit
• Max. current load	5 mA
• Reference	GND
Digital input ports	2 Bits
• 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
• Reference	GND
Digital output port	2 Bits, buffered
• LOW level	< 0.55 V
• HIGH level	> 3.8 V
• Max. current load	20 mA
• Reference	GND

(a) Not as with RTC6 PCIe Board.

(b) With RTC5 boards and RTC6 PCIe Boards, the ANALOG OUT2 signal is also available at the MARKING ON THE FLY socket connector.

LASER Socket Connector (cont'd)

Input ports for external start and stop signals TTL active-LOW, internally connected to +3.3 V by pull-up resistors (4.7 kΩ)

- /START Edge sensitive
- /STOP Level sensitive
- Reference GND

Other signals

- BUSY OUT 5 V, TTL active-HIGH, Max. 10 mA, Reference GND
- +5 V Max. 100 mA
- Reference GND

EXTENSION 1 Socket Connector

Connector

40 pin socket connector with 2.00 mm pitch of the pins^(a).

Output signal level is configurable by solder jumper field A

Digital output port 16 Bits, buffered

- LOW level < 0.4 V
- HIGH level > 2.0 V (3.3 V or 5 V)
- Max. current load ±8 mA
- Reference GND
- LATCH-Signal 3.3 V or 5 V, TTL active-HIGH, 5 µs pulse, max. 10 mA

Digital input port 16 Bit, protected

- 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

Other signals

- BUSY OUT 3.3 V or 5 V, TTL active-HIGH, max. 10 mA
- VCC 3.3 V or 5 V, max. 100 mA
- +5 V Max. 100 mA
- Reference GND

(a) With RTC6 PCIe Boards: 2.54 mm pitch of the pins.

EXT. 2 Socket Connector

Connector	10 pin socket connector with 2.00 mm pitch of the pins. pin (09) ^(a) is configurable by solder jumper field B. pin (08) ^(b) is configurable by solder jumper field C
Digital output port	8 Bits, buffered <ul style="list-style-type: none"> • LOW level < 0.4 V • HIGH level > 2.0 V • Max. current load ± 8 mA • Reference GND • LATCH-Signal 5 V, TTL active-HIGH, 5 μs pulse, max. 10 mA
laser control signals	None ^{(c)(d)}
Other signals	<ul style="list-style-type: none"> • +5 V Max. 100 mA • Reference GND

- (a) pin (09) of RTC6 Ethernet Boards = pin (17) of RTC6 PCIe Boards.
- (b) pin (08) of RTC6 Ethernet Boards = pin (15) of RTC6 PCIe Boards.
- (c) Not as with RTC6 PCIe Board.
- (d) With RTC6 PCIe Boards: LASER1 and LASER2.

MOF Socket Connector^(a)

Connector	14 pin socket connector with 2.00 mm pitch of the pins ^(a)
2 encoder input ports 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 (RS-422) each. HIGH level \geq 2.0 V LOW level \leq 0.8 V $f \leq 4$ MHz
Analog output port	None ^{(b)(c)}
Input ports for external start and stop signals	/START2, /STOP2 (see /START, /STOP of LASER socket connector, page 902)
Other signals	<ul style="list-style-type: none"> • BUSY OUT Identical with BUSY OUT on LASER socket connector • +5 V Max. 100 mA • Reference GND
	(a) With RTC6 PCIe Boards: 16 pin socket connector with 2.54 mm pitch of the pins. Furthermore, the printed label on the board is MARKING ON THE FLY.
	(b) Not as with RTC6 PCIe Board.
	(c) The RTC6 Ethernet Board has no pin for the ANALOG OUT2 signal on its MOF socket connector.

SPI/ANA/UART Socket Connector

With RTC6 Ethernet Boards, the RS232 socket connector pins and McBSP/ANALOG socket connector pins of the RTC6 PCIe Board are integrated into the SPI/ANA/UART socket connector.

Connector 10 pin socket connector with 2.00 mm pitch of the pins.

Analog input ports **ANALOG IN0**,
ANALOG IN1

- Input voltage range 0 V...10 V
- Input impedance $> 5 \text{ k}\Omega^{\text{(a)}}$
- ADC resolution 12 Bit
- Reference GND^(b)

Pins for RS-232

Input port	RxD	• Max. current load	25 mA
	• Voltage range	Max. -25 V...+25 V	
Output port	TxD	• Reference	GND
	• Voltage range	Max. -13 V...+13 V	
Reference	GND		
Baud rate	300...115,200		

Pins for **McBSP**, see also **Section "McBSP Interface"**,
page 83

- Transmitter signal 3.3 V TTL level
- Receiver signal level 3.3 V or 5 V TTL
- **McBSP mode** Single Phase Frame
Single Element per Frame
32 Bits per Element
Data delay **XDelay** bits
Data delay **RDelay** bits
- Reference GND

SPI interface
functionality

(a) *Not* as with RTC6 PCIe Board.

(b) See **Notes**, **page 864**.

STEPPER Socket Connector

Connector 10 pin socket connector with 2.00 mm pitch of the pins.

Signals for controlling up to two stepper motors:

- | | |
|---|--|
| • Output ports
ENABLE1,
ENABLE2,
DIRECTION1,
DIRECTION2 | 5 V, TTL |
| • Output ports
CLOCK1,
CLOCK2 | 5 V, TTL active-HIGH,
5 μs pulse |
| • Input ports
SWITCH1,
SWITCH2 | TTL active-LOW, internally
connected to +3.3 V by
pull-up resistors (10 k Ω) |
| • Max. current load | 25 mA |
| • Reference | GND |

17 Appendix B: The UFP Extention Board

The UFP Extention Board⁽¹⁾ has been developed for RTC6 PCIe Boards⁽²⁾.

The assembly is shown in [Figure 89](#), dimensions and details in [Figure 90](#).

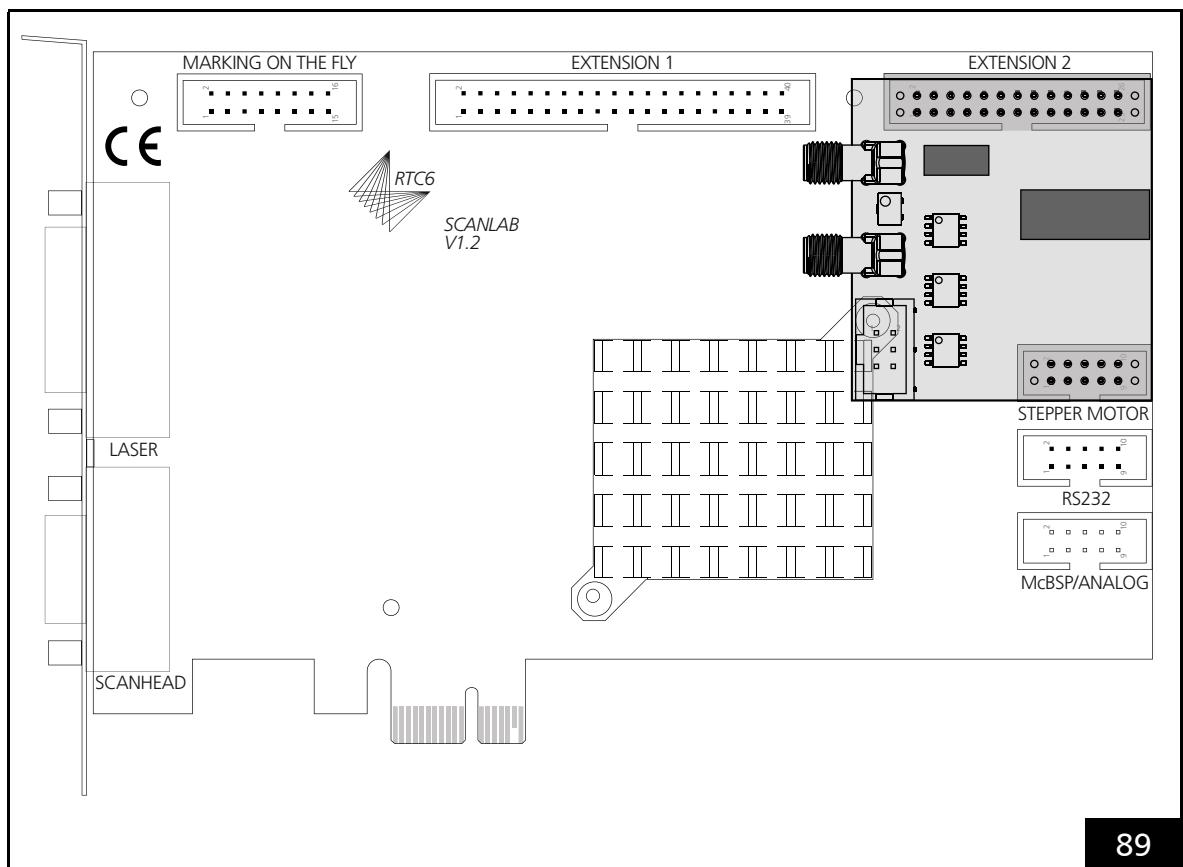
The UFP Extention Board converts 8-bit digital signals into analog voltage values using a fast digital-to-analog converter.

It is recommended, if:

- Analog controlled lasers are operated and
 - Pixel output frequencies ≥ 100 kHz⁽³⁾⁽⁴⁾ are to be achieved (see [set_pixel_line](#))
- (3) [Option "UFP"](#) is mandatory for pixel output frequencies 800 kHz...3,2 MHz.
- (4) The UFP Extention Board also supports pixel output frequencies < 100 kHz, of course.

(1) For example, #137980, #140965.

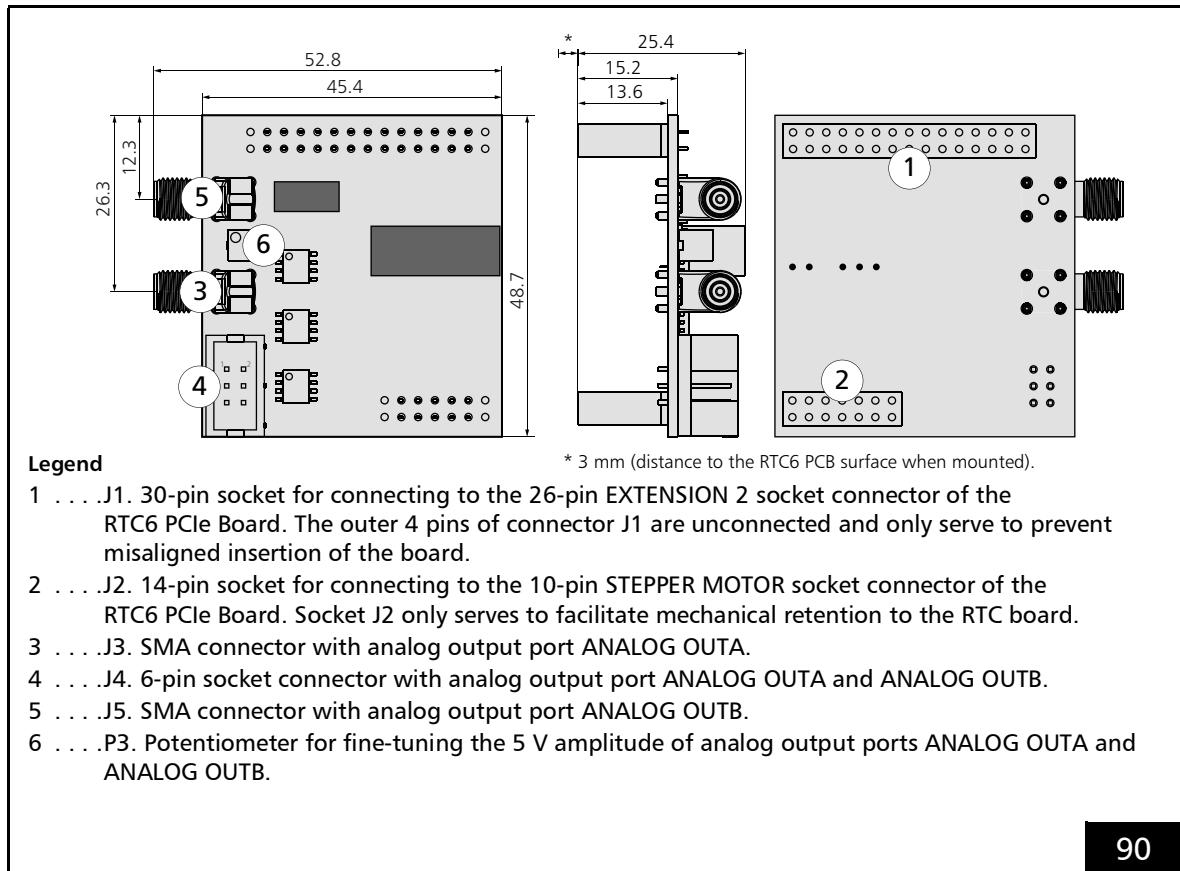
(2) Cannot be uses with RTC6 Ethernet Boards for mechanical reasons.



89

UFP Extention Board: Assembly with the RTC6 PCIe Board.

Note: The [Second Scan Head Slot Cover \(Accessory\)](#) (#115132) cannot be used.



UFPM Extension Board: dimensions and details. All dimensions in mm.

90

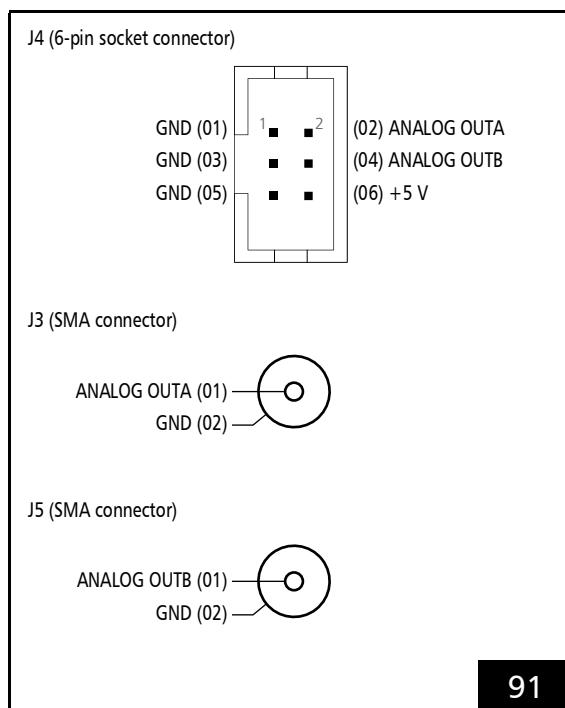
The UFPM Extension Board can be used to control the pixel-to-pixel variation in laser power via analog voltage values with 8 bit resolution.

Output voltage range: 0 V...5 V.

For this purpose, the outputs must be sent to Port = 3 (8-bit digital output port at the EXTENSION 2 socket connector), see [set_pixel_line](#).

For all [Pixel Output Modes](#) including their extensions, the laser pulse duration must be specified before the beginning of the pixel line by [set_laser_pulses](#), [set_laser_pulses_ctrl](#) or [set_laser_timing](#) with at least $1/64 \mu\text{s}$ duration (even if the pulse duration is not used for laser control). Otherwise, no pulses are outputted at LASER1.

The UFPM Extension Board adapts the LATCH signal duration to the pixel frequency (approx. HalfPeriod...5 µs). It synchronously converts the 8-bit digital values to analog values in the range 0 V...5 V and makes them available at its analog output ports ANALOG OUTA and ANALOG OUTB. The connector pin-outs on the UFPM Extension Board are shown in **Figure 91**. Potentiometer P3 lets you fine-tune the exact voltage amplitude, see **Figure 90**.



91

UFPM Extension Board: connector pin-outs.

Software Requirements

- The UFPM Extension Board is supported by default by the RTC6 Software Package package.

Hardware Requirements

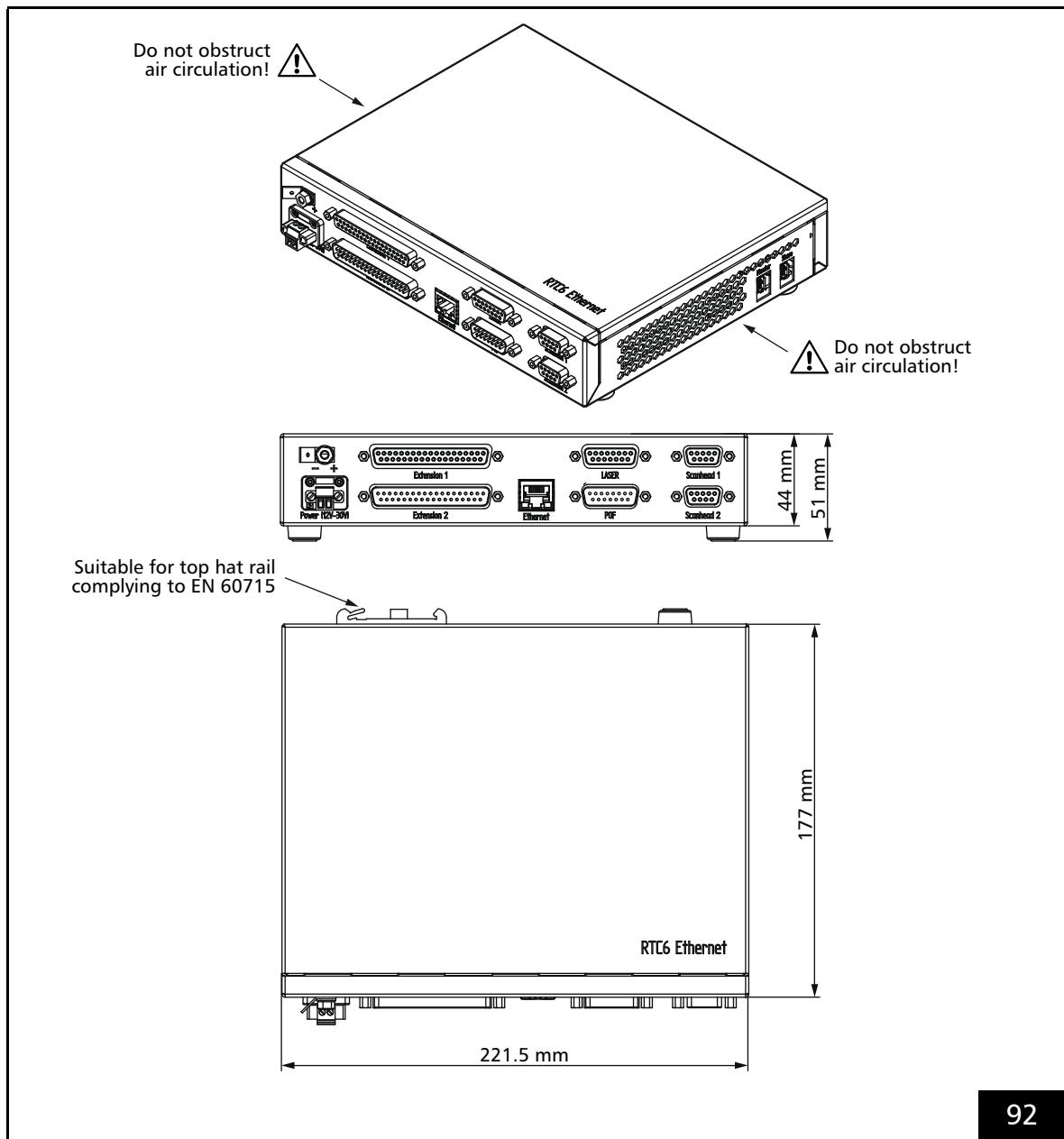
- The pixel values are outputted as 8-bit digital values at the 8-bit digital output of the RTC6 PCIe Board, EXTENSION 2 socket connector. To enable outputting of (complete) pixel values as well as the latch signal at the EXTENSION 2 socket connector, the following solder jumper on the RTC6 PCIe Board must be closed⁽¹⁾:
 - DATA7 in solder jumper field C
(= the DATA7 signal is outputted at pin 15 of the EXTENSION 2 socket connector)
 - LATCH in solder jumper field B
(= the LATCH signal is outputted at pin 17 of the EXTENSION 2 socket connector)

Technical Specifications

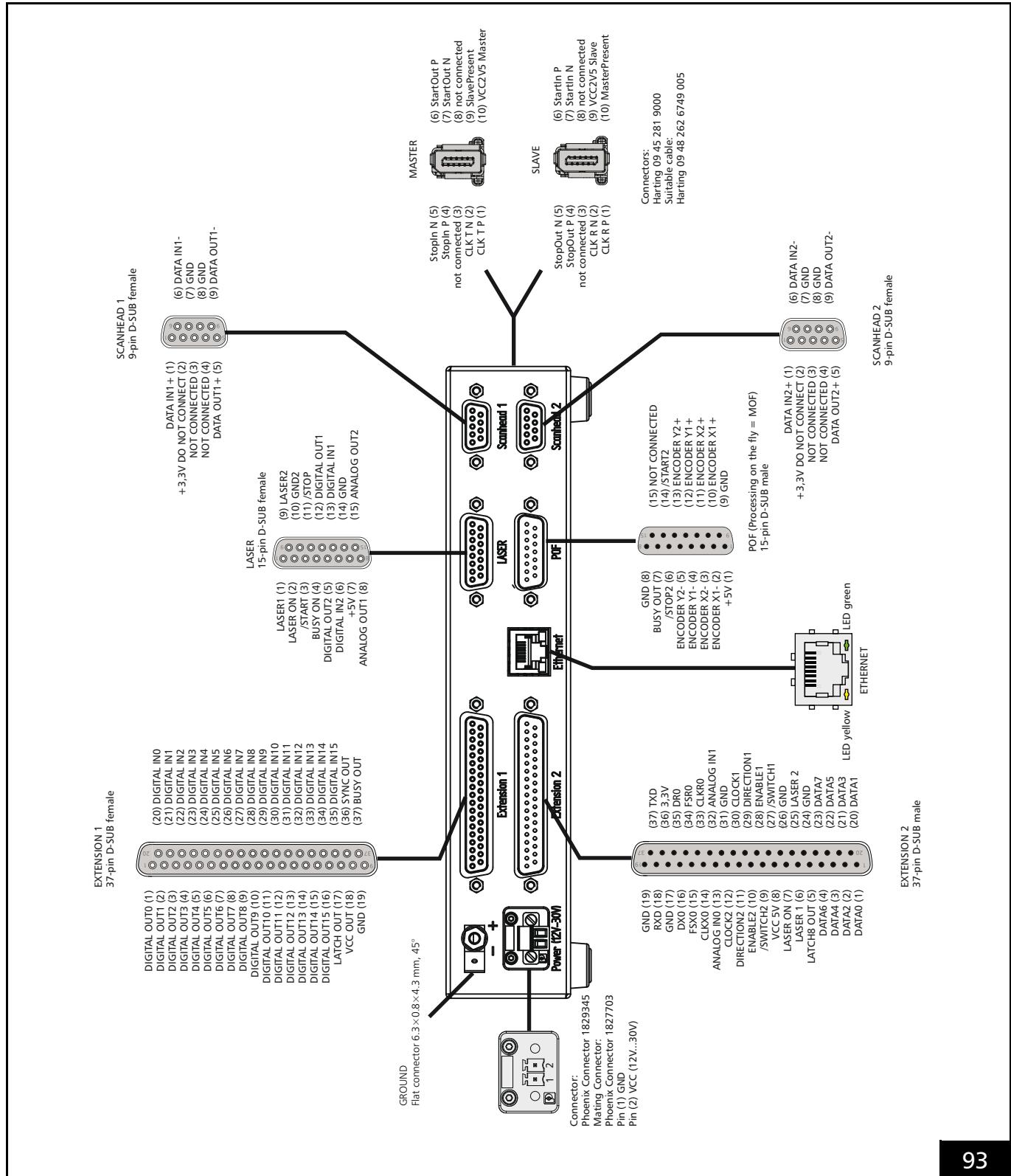
- Dimensions
 - Length 52.8 mm
 - Width 48.7 mm
- Analog output ports ANALOG OUTA, ANALOG OUTB
 - Connectors 1 6-pin socket connector, 2 SMA connectors (coaxial connectors)
 - Output voltage range 0 V...5 V
 - Resolution 8 bits
 - Max. current load 5 mA
 - Reference GND

(1) Corresponds to RTC6 PCIe Boards TYPE n24.

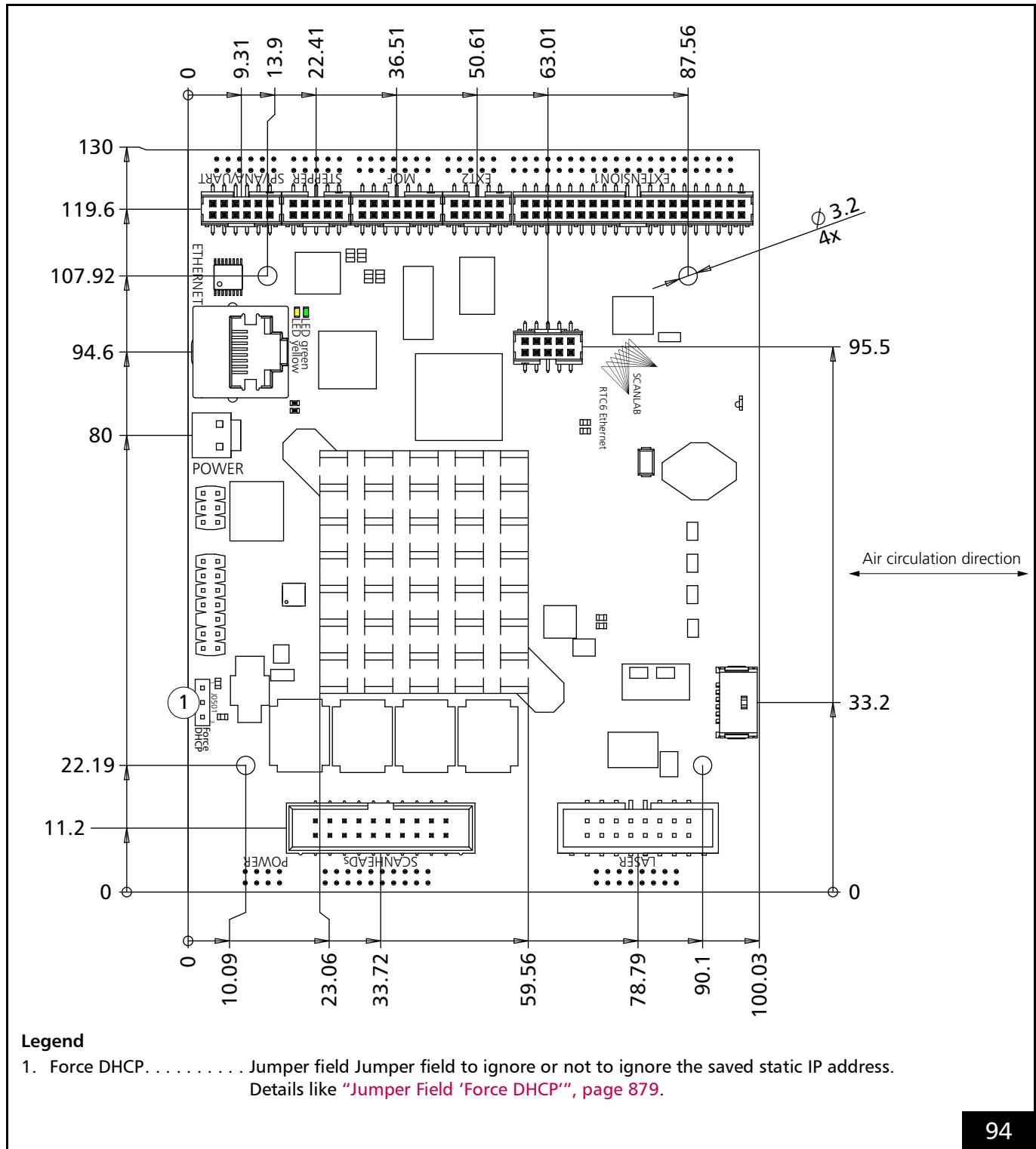
18 Appendix C: The RTC6 EtherBox



RTC6 EtherBox. Dimensions. For example, #148520.



RTC6 EtherBox: pin-outs.



RTC6 EtherBox: Installed RTC6 Ethernet Board. Upper side. Dimensions and connector positions. All dimensions in mm.

18.1 Mounting

Notice!

- The RTC6 board will be irreversibly destroyed if overheated. Make sure that the perforated sides of the RTC6 EtherBox are never closed or covered.
- Use the top-hat rail bracket on the backside, see [Figure 92](#), to hang the RTC6 EtherBox in a top hat rail. alternatively, place the device free.

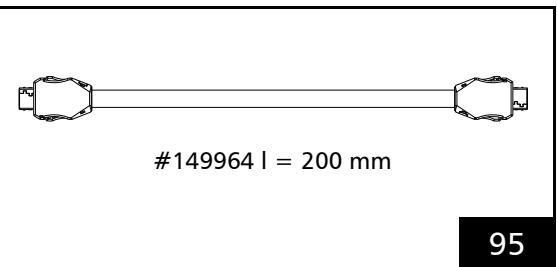
18.2 Cabling



Caution!

- Even when both RTC6 EtherBoxes are connected to the same power supply, different potentials can result of unfavorable cabling and can destroy the boards. To avoid them:
 - Connect one RTC6 EtherBox to the power supply. Then connect the other RTC6 EtherBox by a cable junction to the same power supply.
 - Keep the connection between the RTC6 EtherBoxes as short as possible.
 - First do the cabling of the components correctly, then switch on the power supply.
 - Switch off the power supply before disconnect the cabling.
- Power is supplied via a Phoenix connector, see [Figure 93](#). Use the delivered Phoenix mating connector (manufacturer's article number: 1827703).
- SCANLAB recommends grounding the device. To do so, connect the grounding to the flat connector (6.3×0.8×4.3 mm).

- Connect the PC to the Ethernet connector via an Ethernet cable.
- Connect your desired devices to the respective connector. Observe the pin-outs of the connectors, see [Figure 93](#).
- For the electrical integration of the RTC6 EtherBox, use only the connections on the outside of the housing. Do not use any pins on the built-in RTC6 Ethernet Board (see [Figure 94](#)) that are not connected by SCANLAB to connectors on the housing of the RTC6 EtherBox.
- *Do not* connect the master/slave connector of the RTC6 EtherBox to that of separate RTC6 boards because the pin-outs are different.
- If you supply multiple RTC6 Ethernet Boards/RTC6 EtherBoxes with voltage from only one source, you must implement the cabling as shown in [Figure 88](#).
- If you use several RTC6 EtherBoxes together⁽¹⁾, take care of a correct master/slave connection. A suitable cable is available from SCANLAB, see [Figure 95](#).



SCANLAB Master/Slave connecting cable for RTC6 EtherBoxes.

95

(1) See also [Chapter 6.6.3 "Master/Slave Operation"](#), page 123.



18.3 Installation and Operation

- When installing the RTC6 Ethernet Board, a static IP address must be obtained. For that, you have to set the jumper field **Force DHCP** on the RTC Ethernet board.

- (1) Make sure that the RTC6 EtherBox is disconnected from the power supply.
- (2) Unscrew the two Torx screws on the back of the RTC6 EtherBox and remove the housing cover.
- (3) Set the jumper field '**Force DHCP**', see [Figure 94](#), to "**Force DHCP position**":



"Force DHCP position"

- (4) Put the housing cover back on and screw the two Torx screws from step 2.
- (5) Connect the RTC6 Ethernet Board to the power supply back on again and a network, see also [Chapter 16.3.3 "Connecting to a Network"](#), page 881.
- (6) More information about installation and use of the RTC6 Ethernet Board (inside the RTC6 EtherBox) can be found in the main part of the RTC6 Manual.



18.4 Legal

18.4.1 EU Declaration of Conformity – RTC6 EtherBox



EU-Konformitätserklärung

für das Produkt:

RTC6 EtherBox

Der Hersteller

SCANLAB GmbH, Siemensstr. 2a, 82178 Puchheim, Deutschland

erklärt, dass das genannte Produkt die einschlägigen Harmonisierungsrechtsvorschriften der Union erfüllt:

- **2011/65/EU** - Richtlinie des EUROPÄISCHEN PARLAMENTS UND DES RATES zur Beschränkung der Verwendung bestimmter gefährlicher Stoffe in Elektro- und Elektronikgeräten (RoHS-Richtlinie).

Folgende harmonisierte Normen wurden angewandt:

- **DIN EN IEC 63000:2019-05** - Technische Dokumentation zur Beurteilung von Elektro- und Elektronikgeräten hinsichtlich der Beschränkung gefährlicher Stoffe

Die alleinige Verantwortung für die Ausstellung dieser Konformitätserklärung trägt der Hersteller.

Dieses Dokument zur Kundeninformation wurde elektronisch ausgestellt und ist daher ohne Unterschrift gültig. Eine unterschriebene Ausführung ist Bestandteil der SCANLAB-internen technischen Dokumentation.

EU Declaration of Conformity

for the product:

RTC6 EtherBox

The manufacturer

SCANLAB GmbH, Siemensstr. 2a, 82178 Puchheim, Germany

declares that the product mentioned above complies with the relevant Union harmonization legislation:

- **2011/65/EU** - Directive of the EUROPEAN PARLIAMENT AND OF THE COUNCIL on the restriction of the use of certain hazardous substances in electrical and electronic equipment (RoHS Directive).

The following harmonized standards have been applied:

- **DIN EN IEC 63000:2019-05** - Technical documentation for the assessment of electrical and electronic products with respect to the restriction of hazardous substances

This declaration of conformity is issued under the sole responsibility of the manufacturer.

This document, used for customer information, was issued electronically and is therefore valid without a signature. A signed version is part of the SCANLAB internal technical documentation.

SCANLAB GmbH
Puchheim

2021-07-19

© SCANLAB GmbH – 2021/07



18.4.2 Compliance with FCC Rules

The RTC6 EtherBox 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 RTC6 EtherBox is operated in a commercial environment. The RTC6 EtherBox 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 RTC6 EtherBox 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.

18.5 Technical Specifications – RTC6 EtherBox

Dimensions

See [Figure 93](#).

Weight

1.4 kg

Electrical Specifications

Voltage supply^(a) +12 V...+30 V

Maximum power consumption, no peripherals attached < 12 W

(a) Power supply not included.

19 Appendix D: RTC6 Software Packages – History

RTC6 Software Package Version	DLL (a)	OUT (a)	ETH (a)	RBF (a)	DAT (a)	BIOS (b)	BIOS-ETH (c)	Date
1.3.0	606	606	*(d)	611	601	21	*	2017-09-11
1.3.1	607	607	607	611	601	22	*	2017-11-09
1.3.2	608	608	608	611	601	22	22	2018-01-23
1.3.3	608	608	608	612	601	22	22	2018-04-19
1.4.1	609	609	609	614	601	23	23	2018-08-03
1.4.2	610	610	610	615	602	23	23	2018-11-15
1.4.4	611	611	611	615	603	23	24	2019-03-01
1.5.0	614	614	614	619	603a	23	25	2019-07-26
1.5.2	615	615	615	621	603a	23	25	2019-09-11
1.6.0	616	616	616	622	603a	23	25	2019-11-25
1.6.1	617	617	617	623	603a	23	25	2020-02-07
1.7.0	618	618	618	623	603a	23	26	2020-03-13
1.7.1	618	618	618	623	603a	23	26	2020-03-19
1.7.3	619	619	619	624	603a	23	27	2020-06-19
1.7.4	620	620	620	625	603a	23	27	2020-07-10
1.7.5	621	621	621	625	603a	23	27	2020-07-24
1.7.6	622	622	622	625	603a	23	28	2020-10-02
1.7.7	623	623	623	626	603a	23	28	2020-12-11
1.7.8	624	624	624	627	603a	23	28	2021-01-22
1.7.9	625	625	625	628	603a	23	28	2021-02-05
1.7.10	625	626	626	628	603a	23	28	2021-02-26
1.7.11	626	627	627	629	603a	23	28	2021-04-30
1.7.12	627	628	628	629	603a	23	29	2021-07-09
1.9.0	628	629	629	630	603a	23	30	2021-09-07
1.10.0	629	630	630	631	603a	23	31	2021-10-15
1.11.0	630	631	631	631	603a	23	32	2021-11-12
1.12.0	631	632	632	632	604	23	33	2021-12-22
1.13.0	632	633	633	633	604	23	33	2022-02-18
1.14.1	634	635	635	634	604	23	35	2022-07-22

(a) See Table [page 24](#).

(b) RTC6BIOSOUT_*.out, see [page 33](#).

(c) RTC6BIOSETH_*.out, see [page 33](#).

(d) Cannot be updated by users or version cannot be read out.

20 Appendix E: iDRIVE Scan Systems – Control Commands and Signals Transmitted to RTC Control Boards

- For usage, see [control_command](#).

- Example:

```
control_command( Data = 0501H )
that is, CodeHIGH = 05 (SetMode) and
CodeLOW = 01 requests that the
iDRIVE scan system should change the data type
to be transmitted to "actual position". The
iDRIVE scan system evaluates the request and – if
successful – subsequently transmits the
"actual position" to the RTC-control board
(notation in this manual is "Signal 0501H" in this
case).
```

- The following reference is used in several
RTC manuals and is thus generic.

Code _{HIGH} (hex)	
05	<p>SetMode</p> <p>SetMode selects the data signal to be returned from the scan system for the specified axis. See also Chapter 8.1.2 "Configuring the Data Signal Transmission Behavior of the Scan System", page 212.</p> <p>Each Code_{LOW} parameter value corresponds to a particular data type. Default setting: Code_{LOW} = 00_H (XY2-100 status word). See also "On SetMode", Seite 946. See also "On SetMode, SetControlDefinitionMode and SetEchoMode", Seite 946.</p> <p>iDRIVE scan systems with XY2-100 interface transmit a signed 16-bit value. iDRIVE scan systems with SL2-100 interface transmit an signed 20-bit value. For example, excelliSCAN-scan heads. iDRIVE scan systems with XY2-100 Enhanced interface transmit a signed 16-bit value. With these, the XY2-100 Converter (Accessory) converts (by multiplying by 16) the value to a signed 20-bit value.</p>

Notice! Generic information. May be incomplete or even incorrect for your scan system.

Always consult the dedicated scan system manual!

Module Rev.1.0.3 en-US

Code _{HIGH} (hex)	Code _{LOW} (hex)	Data type transmitted by the scan system																																																															
05	00	<p>As of scan system firmware \geq 2001.</p> <p>XY2-100 status word</p> <table> <tr> <td>16-bit protocol</td> <td>20-bit protocol</td> <td></td> </tr> <tr> <td>Bit #15 (MSB)</td> <td>Bit #19 (MSB)</td> <td>Like [Bit #7 Bit #11]. = 1: The scan system servo control is ready for input data ("PowerOK", actually corresponds to a "ServoOK").</td> </tr> <tr> <td>Bit #14</td> <td>Bit #18</td> <td>Like [Bit #6 Bit #10]. = 1: The galvanometer scanner temperature within optimal range ("TempOK").</td> </tr> <tr> <td>Bit #13</td> <td>Bit #17</td> <td>= 1.</td> </tr> <tr> <td>Bit #12</td> <td>Bit #16</td> <td>Like [Bit #4 Bit #8]. = 1: The y axis position errors are within allowed range (PosAck signal).</td> </tr> <tr> <td>Bit #11</td> <td>Bit #15</td> <td>Like [Bit #3 Bit #7]. = 1: The x axis position errors are within allowed range (PosAck signal).</td> </tr> <tr> <td>Bit #10</td> <td>Bit #14</td> <td>Like [Bit #2 Bit #6]. = 1. Only intelliWELD: Reserved. For scan systems with sensors for automatic self calibration: Reserved.</td> </tr> <tr> <td>Bit #9</td> <td>Bit #13</td> <td>= 0.</td> </tr> <tr> <td>Bit #8</td> <td>Bit #12</td> <td>= 1.</td> </tr> <tr> <td>Bit #7</td> <td>Bit #11</td> <td>Like [Bit #15 Bit #19].</td> </tr> <tr> <td>Bit #6</td> <td>Bit #10</td> <td>Like [Bit #14 Bit #18].</td> </tr> <tr> <td>Bit #5</td> <td>Bit #9</td> <td>= 1.</td> </tr> <tr> <td>Bit #4</td> <td>Bit #8</td> <td>Like [Bit #12 Bit #16].</td> </tr> <tr> <td>Bit #3</td> <td>Bit #7</td> <td>Like [Bit #11 Bit #15].</td> </tr> <tr> <td>Bit #2</td> <td>Bit #6</td> <td>Like [Bit #10 Bit #14].</td> </tr> <tr> <td>Bit #1</td> <td>Bit #5</td> <td>= 0.</td> </tr> <tr> <td>Bit #0</td> <td>Bit #4</td> <td>= 1.</td> </tr> <tr> <td>–</td> <td>Bit #3</td> <td>= 0.</td> </tr> <tr> <td>–</td> <td>Bit #2</td> <td>= 0.</td> </tr> <tr> <td>–</td> <td>Bit #1</td> <td>= 0.</td> </tr> <tr> <td>–</td> <td>Bit #0</td> <td>= 0.</td> </tr> </table>	16-bit protocol	20-bit protocol		Bit #15 (MSB)	Bit #19 (MSB)	Like [Bit #7 Bit #11]. = 1: The scan system servo control is ready for input data ("PowerOK", actually corresponds to a "ServoOK").	Bit #14	Bit #18	Like [Bit #6 Bit #10]. = 1: The galvanometer scanner temperature within optimal range ("TempOK").	Bit #13	Bit #17	= 1.	Bit #12	Bit #16	Like [Bit #4 Bit #8]. = 1: The y axis position errors are within allowed range (PosAck signal).	Bit #11	Bit #15	Like [Bit #3 Bit #7]. = 1: The x axis position errors are within allowed range (PosAck signal).	Bit #10	Bit #14	Like [Bit #2 Bit #6]. = 1. Only intelliWELD: Reserved. For scan systems with sensors for automatic self calibration: Reserved.	Bit #9	Bit #13	= 0.	Bit #8	Bit #12	= 1.	Bit #7	Bit #11	Like [Bit #15 Bit #19].	Bit #6	Bit #10	Like [Bit #14 Bit #18].	Bit #5	Bit #9	= 1.	Bit #4	Bit #8	Like [Bit #12 Bit #16].	Bit #3	Bit #7	Like [Bit #11 Bit #15].	Bit #2	Bit #6	Like [Bit #10 Bit #14].	Bit #1	Bit #5	= 0.	Bit #0	Bit #4	= 1.	–	Bit #3	= 0.	–	Bit #2	= 0.	–	Bit #1	= 0.	–	Bit #0	= 0.
16-bit protocol	20-bit protocol																																																																
Bit #15 (MSB)	Bit #19 (MSB)	Like [Bit #7 Bit #11]. = 1: The scan system servo control is ready for input data ("PowerOK", actually corresponds to a "ServoOK").																																																															
Bit #14	Bit #18	Like [Bit #6 Bit #10]. = 1: The galvanometer scanner temperature within optimal range ("TempOK").																																																															
Bit #13	Bit #17	= 1.																																																															
Bit #12	Bit #16	Like [Bit #4 Bit #8]. = 1: The y axis position errors are within allowed range (PosAck signal).																																																															
Bit #11	Bit #15	Like [Bit #3 Bit #7]. = 1: The x axis position errors are within allowed range (PosAck signal).																																																															
Bit #10	Bit #14	Like [Bit #2 Bit #6]. = 1. Only intelliWELD: Reserved. For scan systems with sensors for automatic self calibration: Reserved.																																																															
Bit #9	Bit #13	= 0.																																																															
Bit #8	Bit #12	= 1.																																																															
Bit #7	Bit #11	Like [Bit #15 Bit #19].																																																															
Bit #6	Bit #10	Like [Bit #14 Bit #18].																																																															
Bit #5	Bit #9	= 1.																																																															
Bit #4	Bit #8	Like [Bit #12 Bit #16].																																																															
Bit #3	Bit #7	Like [Bit #11 Bit #15].																																																															
Bit #2	Bit #6	Like [Bit #10 Bit #14].																																																															
Bit #1	Bit #5	= 0.																																																															
Bit #0	Bit #4	= 1.																																																															
–	Bit #3	= 0.																																																															
–	Bit #2	= 0.																																																															
–	Bit #1	= 0.																																																															
–	Bit #0	= 0.																																																															

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!

Module Rev.1.0.3 en-US

Code _{HIGH} (hex)	Code _{LOW} (hex)	Data type transmitted by the scan system		
05	01	As of scan system firmware ≥ 2001. Actual position (angular position of galvanometer scanners).		
		16-bit protocol	20-bit protocol	
		Bit #0 ... Bit #15	Bit #0 ... Bit #19	Unit with 16-bit protocol: In bits. Value range with 16-bit protocol: [-32.768...+32.767]. Unit with 20-bit protocol: In bits. Value range with 20-bit protocol: -524.288...+524.287.
05	02	As of scan system firmware ≥ 2001. Set position (angular position of galvanometer scanners). As of scan system firmware ≥ 5050. With excelliSCAN scan heads: Precalculated set position (angular position of galvanometer scanners). Refer to "excelliSCAN Scan Heads – Functional Principle of SCANAhead Servo Control and Operation by RTC6 Boards" Manual, Chapter 3.1.7 "excelliSCAN Timing Diagram (Scan Head with SCANAhead Control)", page 16 and Figure 4.		
		16-bit protocol	20-bit protocol	
		Bit #0 ... Bit #15	Bit #0 ... Bit #19	Unit with 16-bit protocol: In bits. Value range with 16-bit protocol: [-32.768...+32.767]. Unit with 20-bit protocol: In bits. Value range with 20-bit protocol: -524.288...+524.287.

Notice! Generic information. May be incomplete or even incorrect for your scan system.

Always consult the dedicated scan system manual!

Module Rev.1.0.3 en-US



Code _{HIGH} (hex)	Code _{LOW} (hex)	Data type transmitted by the scan system		
05	03	As of scan system firmware ≥ 2001. Position error (= set position - actual position).		
		16-bit protocol	20-bit protocol	
		Bit #0 ... Bit #15	Bit #0 ... Bit #19	Unit with 16-bit protocol: In bits. Value range with 16-bit protocol: [-32.768...+32.767]. Unit with 20-bit protocol: In bits. Value range with 20-bit protocol: -524.288...+524.287.
05	04	As of scan system firmware ≥ 2001. Actual current (output stage current).		
		16-bit protocol	20-bit protocol	
		Bit #0 ... Bit #15	Bit #0 ... Bit #19	Unit with 16-bit protocol: In mA. Value range with 16-bit protocol: [-32.768...+32.767]. Unit with 20-bit protocol: In 16 ⁻¹ mA. Value range with 20-bit protocol: -524.288...+524.287.

Notice! Generic information. May be incomplete or even incorrect for your scan system.

Always consult the dedicated scan system manual!

Module Rev.1.0.3 en-US

Code _{HIGH} (hex)	Code _{LOW} (hex)	Data type transmitted by the scan system		
05	05	As of scan system firmware ≥ 2001. Relative galvanometer scanner control. Not intellicube.		
		16-bit protocol	20-bit protocol	
		Bit #0 ... Bit #15	Bit #0 ... Bit #19	Unit with 16-bit protocol: In per mille. Value range with 16-bit protocol: [-1,000...+1,000]. Unit with 20-bit protocol: In 10^{-1} per mille. Value range with 20-bit protocol: [-16,000...+16,000]. The return value 16 entspricht 1%.
05	06	As of scan system firmware ≥ 2001. Actual velocity (angular velocity).		
		16-bit protocol	20-bit protocol	
		Bit #0 ... Bit #15	Bit #0 ... Bit #19	Unit with 16-bit protocol: In Bit/ms. Value range with 16-bit protocol: [-32.768...+32.767]. Unit with 20-bit protocol: In Bit/ms. Value range with 20-bit protocol: -524.288...+524.287.
05	07	Reserved.		
05	08	Reserved.		
05	09	Reserved.		
05	10	Reserved.		
05	11	Reserved.		

Notice! Generic information. May be incomplete or even incorrect for your scan system.

Always consult the dedicated scan system manual!

Module Rev.1.0.3 en-US

Code _{HIGH} (hex)	Code _{LOW} (hex)	Data type transmitted by the scan system			
05	12	As of scan system firmware ≥ 2001. Only intelliWELD with varioSCAN de FCi. Not: intelliWELD with varioSCAN FC: Actual z axis position. Can only be read out from the channel of the scan head connector to which the z axis is connected.			
		16-bit protocol	20-bit protocol		
		Bit #0 ... Bit #15	Bit #0 ... Bit #19	Unit with 16-bit protocol: In bits. Value range with 16-bit protocol: [-32.768...+32.767]. Unit with 20-bit protocol: In bits. Value range with 20-bit protocol: -524.288...+524.287.	
05	13	Reserved.			
05	14	As of scan system firmware ≥ 2001. Temperature of galvanometer scanner.			
		16-bit protocol	20-bit protocol		
		Bit #0 ... Bit #15	Bit #0 ... Bit #19	Unit with 16-bit protocol: In 10^{-1} °C. Value range with 16-bit protocol: [-32.768...+32.767]. Unit with 20-bit protocol: In 160^{-1} °C. Value range with 20-bit protocol: -524.288...+524.287.	

Notice! Generic information. May be incomplete or even incorrect for your scan system.

Always consult the dedicated scan system manual!

Module Rev.1.0.3 en-US



Code _{HIGH} (hex)	Code _{LOW} (hex)	Data type transmitted by the scan system		
05	15	As of scan system firmware \geq 2001. Servo board temperature. As of scan system firmware \geq 5050. Axis 1 (X-axis): Temperature of the servo board (ISB). Axis 2 (Y-axis): Temperature of the servo add-on board for evaluating the encoder signals.		
		16-bit protocol	20-bit protocol	
		Bit #0 ... Bit #15	Bit #0 ... Bit #19	Unit with 16-bit protocol: In 10^{-1} °C. Value range with 16-bit protocol: [-32.768...+32.767]. Unit with 20-bit protocol: In 160^{-1} °C. Value range with 20-bit protocol: -524.288...+524.287.
05	16	As of scan system firmware \geq 2001. AGC voltage (position detector supply voltage).		
		16-bit protocol	20-bit protocol	
		Bit #0 ... Bit #15	Bit #0 ... Bit #19	Unit with 16-bit protocol: In 10^{-2} V. Value range with 16-bit protocol: [-32.768...+32.767]. Unit with 20-bit protocol: In 1600^{-1} V. Value range with 20-bit protocol: -524.288...+524.287.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
 Always consult the dedicated scan system manual!
 Module Rev.1.0.3 en-US



Code _{HIGH} (hex)	Code _{LOW} (hex)	Data type transmitted by the scan system		
05	17	As of scan system firmware \geq 2001. DSP core supply voltage.		
		16-bit protocol	20-bit protocol	
		Bit #0 ... Bit #15	Bit #0 ... Bit #19	Unit with 16-bit protocol: In 10^{-2} V. Value range with 16-bit protocol: [-32.768...+32.767]. Unit with 20-bit protocol: In 1600^{-1} V. Value range with 20-bit protocol: -524.288...+524.287.
05	18	As of scan system firmware \geq 2001. DSP IO voltage.		
		16-bit protocol	20-bit protocol	
		Bit #0 ... Bit #15	Bit #0 ... Bit #19	Unit with 16-bit protocol: In 10^{-2} V. Value range with 16-bit protocol: [-32.768...+32.767]. Unit with 20-bit protocol: In 1600^{-1} V. Value range with 20-bit protocol: -524.288...+524.287.

Notice! Generic information. May be incomplete or even incorrect for your scan system.

Always consult the dedicated scan system manual!

Module Rev.1.0.3 en-US



Code _{HIGH} (hex)	Code _{LOW} (hex)	Data type transmitted by the scan system		
05	19	As of scan system firmware \geq 2001. Analog section voltage.		
		16-bit protocol	20-bit protocol	
		Bit #0 ... Bit #15	Bit #0 ... Bit #19	Unit with 16-bit protocol: In 10^{-2} V. Value range with 16-bit protocol: [-32.768...+32.767]. Unit with 20-bit protocol: In 1600^{-1} V. Value range with 20-bit protocol: -524.288...+524.287.
05	1A	As of scan system firmware \geq 2001. Analog-digital converter supply voltage.		
		16-bit protocol	20-bit protocol	
		Bit #0 ... Bit #15	Bit #0 ... Bit #19	Unit with 16-bit protocol: In 10^{-2} V. Value range with 16-bit protocol: [-32.768...+32.767]. Unit with 20-bit protocol: In 1600^{-1} V. Value range with 20-bit protocol: -524.288...+524.287.

Notice! Generic information. May be incomplete or even incorrect for your scan system.

Always consult the dedicated scan system manual!

Module Rev.1.0.3 en-US

Code _{HIGH} (hex)	Code _{LOW} (hex)	Data type transmitted by the scan system		
05	1B	As of scan system firmware \geq 2001. AGC current (PD supply current).		
		16-bit protocol	20-bit protocol	
		Bit #0 ... Bit #15	Bit #0 ... Bit #19	Unit with 16-bit protocol: In mA. Value range with 16-bit protocol: [-32.768...+32.767]. Unit with 20-bit protocol: In 16^{-1} mA. Value range with 20-bit protocol: -524.288...+524.287.
05	1C	Reserved.		
		16-bit protocol	20-bit protocol	
		Bit #0 ... Bit #15	Bit #0 ... Bit #19	Reserved.
05	1D	As of scan system firmware \geq 2001. Relevant for scan systems with galvanometer scanner heating. Relative heating output of the corresponding galvanometer scanner heater. Not relevant for intellicube. Not relevant for dynAXIS XS. Not relevant for scan systems with water-cooled galvanometer scanners.		
		16-bit protocol	20-bit protocol	
		Bit #0 ... Bit #15	Bit #4 ... Bit #19	Unit with 16-bit protocol: In per mille. Value range with 16-bit protocol: [0...1,000]. Unit with 20-bit protocol: In per mille. Value range with 20-bit protocol: [0...1,000].
		-	Bit #0 ... Bit #3	= 0.

Notice! Generic information. May be incomplete or even incorrect for your scan system.

Always consult the dedicated scan system manual!

Module Rev.1.0.3 en-US

Code _{HIGH} (hex)	Code _{LOW} (hex)	Data type transmitted by the scan system		
05	1E	As of scan system firmware ≥ 2001. Serial number (lower 16 bits).		
		16-bit protocol	20-bit protocol	
		Bit #0 ... Bit #15	Bit #4 ... Bit #19	Unit with 16-bit protocol: None. Value range with 16-bit protocol: [0...65,535]. Unit with 20-bit protocol: None. Value range with 20-bit protocol: [0...65,535].
		—	Bit #0 ... Bit #3	= 0.
05	1F	As of scan system firmware ≥ 2001. Serial number (higher 16 bits).		
		16-bit protocol	20-bit protocol	
		Bit #0 ... Bit #15	Bit #4 ... Bit #19	Unit with 16-bit protocol: None. Value range with 16-bit protocol: [0...65,535]. Unit with 20-bit protocol: None. Value range with 20-bit protocol: [0...65,535].
		—	Bit #0 ... Bit #3	= 0.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!

Module Rev.1.0.3 en-US

Code _{HIGH} (hex)	Code _{LOW} (hex)	Data type transmitted by the scan system		
05	20	As of scan system firmware ≥ 2001. Article number (lower 16 bits).		
		16-bit protocol	20-bit protocol	
		Bit #0	Bit #4	Unit with 16-bit protocol: None. Value range with 16-bit protocol: [0...65,535].
		Unit with 20-bit protocol: None.
		Bit #15	Bit #19	Value range with 20-bit protocol: [0...65,535].
		–	Bit #0	= 0.
			...	
			Bit #3	
05	21	As of scan system firmware ≥ 2001. Article number (higher 16 bits).		
		16-bit protocol	20-bit protocol	
		Bit #0	Bit #4	Unit with 16-bit protocol: None. Value range with 16-bit protocol: [0...65,535].
		Unit with 20-bit protocol: None.
		Bit #15	Bit #19	Value range with 20-bit protocol: [0...65,535].
		–	Bit #0	= 0.
			...	
			Bit #3	

Notice! Generic information. May be incomplete or even incorrect for your scan system.
 Always consult the dedicated scan system manual!

Module Rev.1.0.3 en-US



Code _{HIGH} (hex)	Code _{LOW} (hex)	Data type transmitted by the scan system											
05	22	As of scan system firmware \geq 2001. Version number of scan system firmware. <table border="1"> <tr> <td>16-bit protocol</td> <td>20-bit protocol</td> <td></td> </tr> <tr> <td>Bit #0 ... Bit #15</td> <td>Bit #4 ... Bit #19</td> <td>Unit with 16-bit protocol: None. Value range with 16-bit protocol: [0...65,535]. Unit with 20-bit protocol: None. Value range with 20-bit protocol: [0...65,535].</td> </tr> <tr> <td>-</td> <td>Bit #0 ... Bit #3</td> <td>= 0.</td> </tr> </table>			16-bit protocol	20-bit protocol		Bit #0 ... Bit #15	Bit #4 ... Bit #19	Unit with 16-bit protocol: None. Value range with 16-bit protocol: [0...65,535]. Unit with 20-bit protocol: None. Value range with 20-bit protocol: [0...65,535].	-	Bit #0 ... Bit #3	= 0.
16-bit protocol	20-bit protocol												
Bit #0 ... Bit #15	Bit #4 ... Bit #19	Unit with 16-bit protocol: None. Value range with 16-bit protocol: [0...65,535]. Unit with 20-bit protocol: None. Value range with 20-bit protocol: [0...65,535].											
-	Bit #0 ... Bit #3	= 0.											
05	23	As of scan system firmware \geq 2001. Calibration angle. Mechanical scan angle (\pm) for 96% of the maximum or minimum control value. With 16-bit protocol ± 31457 bit. With 20-bit protocol ± 503316 bit. <table border="1"> <tr> <td>16-bit protocol</td> <td>20-bit protocol</td> <td></td> </tr> <tr> <td>Bit #0 ... Bit #15</td> <td>Bit #4 ... Bit #19</td> <td>Unit with 16-bit protocol: In 10^{-3} degrees. Value range with 16-bit protocol: [0...65,535]. Unit with 20-bit protocol: In 10^{-3} degrees. Value range with 20-bit protocol: [0...65,535].</td> </tr> <tr> <td>-</td> <td>Bit #0 ... Bit #3</td> <td>= 0.</td> </tr> </table>			16-bit protocol	20-bit protocol		Bit #0 ... Bit #15	Bit #4 ... Bit #19	Unit with 16-bit protocol: In 10^{-3} degrees. Value range with 16-bit protocol: [0...65,535]. Unit with 20-bit protocol: In 10^{-3} degrees. Value range with 20-bit protocol: [0...65,535].	-	Bit #0 ... Bit #3	= 0.
16-bit protocol	20-bit protocol												
Bit #0 ... Bit #15	Bit #4 ... Bit #19	Unit with 16-bit protocol: In 10^{-3} degrees. Value range with 16-bit protocol: [0...65,535]. Unit with 20-bit protocol: In 10^{-3} degrees. Value range with 20-bit protocol: [0...65,535].											
-	Bit #0 ... Bit #3	= 0.											

Notice! Generic information. May be incomplete or even incorrect for your scan system.
 Always consult the dedicated scan system manual!
 Module Rev.1.0.3 en-US

Code _{HIGH} (hex)	Code _{LOW} (hex)	Data type transmitted by the scan system		
05	24	As of scan system firmware ≥ 2001. Aperture.		
		16-bit protocol	20-bit protocol	
		Bit #0 ... Bit #15	Bit #4 ... Bit #19	Unit with 16-bit protocol: In mm. Value range with 16-bit protocol: [0...65,535]. Unit with 20-bit protocol: In mm. Value range with 20-bit protocol: [0...65,535].
		–	Bit #0 ... Bit #3	= 0.
		As of scan system firmware ≥ 2001. Wavelength.		
05	25	16-bit protocol	20-bit protocol	
		Bit #0 ... Bit #15	Bit #4 ... Bit #19	Unit with 16-bit protocol: In nm. Value range with 16-bit protocol: [0...65,535]. Unit with 20-bit protocol: In nm. Value range with 20-bit protocol: [0...65,535].
		–	Bit #0 ... Bit #3	= 0.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!

Module Rev.1.0.3 en-US



Code _{HIGH} (hex)	Code _{LOW} (hex)	Data type transmitted by the scan system		
05	26	As of scan system firmware \geq 2078. Tuning number.		
		16-bit protocol	20-bit protocol	
		Bit #8	Bit #12	Start setting.
		
		Bit #15	Bit #19	
		Bit #0	Bit #4	Current setting.
		
		Bit #7	Bit #11	
		–	Bit #0	= 0.
			...	
			Bit #3	

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.3 en-US



Code _{HIGH} (hex)	Code _{LOW} (hex)	Data type transmitted by the scan system																																
05	27	As of scan system firmware \geq 2078. Only after <code>control_command(Data = 17FF_H)</code> : number of the temporarily stored data type. <table border="1" style="margin-left: 20px;"> <tr> <td>16-bit protocol</td> <td>20-bit protocol</td> <td></td> </tr> <tr> <td>Bit #8</td> <td>Bit #12</td> <td>= 0.</td> </tr> <tr> <td>...</td> <td>...</td> <td></td> </tr> <tr> <td>Bit #15</td> <td>Bit #19</td> <td></td> </tr> <tr> <td>Bit #0</td> <td>Bit #4</td> <td>Cached setting.</td> </tr> <tr> <td>...</td> <td>...</td> <td></td> </tr> <tr> <td>Bit #7</td> <td>Bit #11</td> <td></td> </tr> <tr> <td>–</td> <td>Bit #0</td> <td>= 0.</td> </tr> <tr> <td></td> <td>...</td> <td></td> </tr> <tr> <td></td> <td>Bit #3</td> <td></td> </tr> </table>			16-bit protocol	20-bit protocol		Bit #8	Bit #12	= 0.		Bit #15	Bit #19		Bit #0	Bit #4	Cached setting.		Bit #7	Bit #11		–	Bit #0	= 0.		...			Bit #3	
16-bit protocol	20-bit protocol																																	
Bit #8	Bit #12	= 0.																																
...	...																																	
Bit #15	Bit #19																																	
Bit #0	Bit #4	Cached setting.																																
...	...																																	
Bit #7	Bit #11																																	
–	Bit #0	= 0.																																
	...																																	
	Bit #3																																	
05	28	Reserved.																																
05	29	Reserved.																																

Notice! Generic information. May be incomplete or even incorrect for your scan system.

Always consult the dedicated scan system manual!

Module Rev.1.0.3 en-US

Code _{HIGH} (hex)	Code _{LOW} (hex)	Data type transmitted by the scan system																																																									
05	2A	<p>As of scan system firmware \geq 2001.</p> <p>Stop event code.</p> <p>Value range with 16-bit protocol: [0...65,535].</p> <p>Value range with 20-bit protocol: [0...65,535].</p>																																																									
		<table border="1"> <thead> <tr> <th>16-bit protocol (hex)</th><th>20-bit protocol (hex)</th><th></th></tr> </thead> <tbody> <tr><td>0001</td><td>00010</td><td>The galvanometer scanner has reached a critical edge position.</td></tr> <tr><td>0002</td><td>00020</td><td>AD converter error.</td></tr> <tr><td>0003</td><td>00030</td><td>Temperature in scan system above max. allowed value.</td></tr> <tr><td>0004</td><td>00040</td><td>External power supply voltages have dropped below the allowed value.</td></tr> <tr><td>0005</td><td>00050</td><td>Flags are not valid.</td></tr> <tr><td>0006</td><td>00060</td><td>Reserved.</td></tr> <tr><td>...</td><td>...</td><td>...</td></tr> <tr><td>000C</td><td>000C0</td><td>Reserved.</td></tr> <tr><td>000D</td><td>000D0</td><td>Reserved.</td></tr> <tr><td>000E</td><td>000E0</td><td>Position Acknowledge time out (set position not reached for long time).</td></tr> <tr><td>000F</td><td>000F0</td><td>Reserved.</td></tr> <tr><td>0014</td><td>00140</td><td>Reserved.</td></tr> <tr><td>0015</td><td>00150</td><td>Reserved.</td></tr> <tr><td>0016</td><td>00160</td><td>Reserved.</td></tr> <tr><td>0017</td><td>00170</td><td>Reserved.</td></tr> <tr><td>0018</td><td>00180</td><td>Reserved.</td></tr> <tr><td>0019</td><td>00190</td><td>Reserved.</td></tr> <tr><td>001A</td><td>001A0</td><td>Reserved.</td></tr> </tbody> </table>	16-bit protocol (hex)	20-bit protocol (hex)		0001	00010	The galvanometer scanner has reached a critical edge position.	0002	00020	AD converter error.	0003	00030	Temperature in scan system above max. allowed value.	0004	00040	External power supply voltages have dropped below the allowed value.	0005	00050	Flags are not valid.	0006	00060	Reserved.	000C	000C0	Reserved.	000D	000D0	Reserved.	000E	000E0	Position Acknowledge time out (set position not reached for long time).	000F	000F0	Reserved.	0014	00140	Reserved.	0015	00150	Reserved.	0016	00160	Reserved.	0017	00170	Reserved.	0018	00180	Reserved.	0019	00190	Reserved.	001A	001A0	Reserved.
16-bit protocol (hex)	20-bit protocol (hex)																																																										
0001	00010	The galvanometer scanner has reached a critical edge position.																																																									
0002	00020	AD converter error.																																																									
0003	00030	Temperature in scan system above max. allowed value.																																																									
0004	00040	External power supply voltages have dropped below the allowed value.																																																									
0005	00050	Flags are not valid.																																																									
0006	00060	Reserved.																																																									
...																																																									
000C	000C0	Reserved.																																																									
000D	000D0	Reserved.																																																									
000E	000E0	Position Acknowledge time out (set position not reached for long time).																																																									
000F	000F0	Reserved.																																																									
0014	00140	Reserved.																																																									
0015	00150	Reserved.																																																									
0016	00160	Reserved.																																																									
0017	00170	Reserved.																																																									
0018	00180	Reserved.																																																									
0019	00190	Reserved.																																																									
001A	001A0	Reserved.																																																									

Notice! Generic information. May be incomplete or even incorrect for your scan system.

Always consult the dedicated scan system manual!

Module Rev.1.0.3 en-US

Code _{HIGH} (hex)	Code _{LOW} (hex)	Data type transmitted by the scan system		
05	2B	Reserved.		
05	2C	Reserved.		
05	2D	Reserved.		
05	2E	Reserved.		
05	2F	As of scan-system firmware ≥ 2061. Running time (seconds).		
		16-bit protocol	20-bit protocol	
		Bit #0 ... Bit #15	Bit #4 ... Bit #19	Unit with 16-bit protocol: In s. Value range with 16-bit protocol: [0...59]. Unit with 20-bit protocol: In s. Value range with 20-bit protocol: [0...59].
		—	Bit #0 ... Bit #3	= 0.
		As of scan-system firmware ≥ 2061. Running time (minutes).		
		16-bit protocol	20-bit protocol	
05	30	Bit #0 ... Bit #15	Bit #4 ... Bit #19	Unit with 16-bit protocol: In min. Value range with 16-bit protocol: [0...59]. Unit with 20-bit protocol: In min. Value range with 20-bit protocol: [0...59].
		—	Bit #0 ... Bit #3	= 0.

Notice! Generic information. May be incomplete or even incorrect for your scan system.

Always consult the dedicated scan system manual!

Module Rev.1.0.3 en-US

Code _{HIGH} (hex)	Code _{LOW} (hex)	Data type transmitted by the scan system		
05	31	As of scan-system firmware ≥ 2061. Running time (hours).		
		16-bit protocol	20-bit protocol	
		Bit #0 ... Bit #15	Bit #4 ... Bit #19	Unit with 16-bit protocol: In h. Value range with 16-bit protocol: [0...23]. Unit with 20-bit protocol: In h. Value range with 20-bit protocol: [0...23].
		–	Bit #0 ... Bit #3	= 0.
		As of scan-system firmware ≥ 2061. Running time (days).		
05	32	16-bit protocol	20-bit protocol	
		Bit #0 ... Bit #15	Bit #4 ... Bit #19	Unit with 16-bit protocol: In d. Value range with 16-bit protocol: [0...65,535]. Unit with 20-bit protocol: In d. Value range with 20-bit protocol: [0...65,535].
		–	Bit #0 ... Bit #3	= 0.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!

Module Rev.1.0.3 en-US

Code _{HIGH} (hex)	Code _{LOW} (hex)	Data type transmitted by the scan system		
05	33	As of scan system firmware \geq 206x. Only intelliWELD: 3.3 V sensor board operating voltage.		
		16-bit protocol	20-bit protocol	
		Bit #0 ... Bit #15	Bit #4 ... Bit #19	Unit with 16-bit protocol: In 10^{-3} V. Value range with 16-bit protocol: [0...7,000]. Unit with 20-bit protocol: In 10^{-3} V. Value range with 20-bit protocol: [0...7,000].
		–	Bit #0 ... Bit #3	= 0.
		As of scan system firmware \geq 206x. Only intelliWELD: Sensor board operating temperature.		
05	34	16-bit protocol	20-bit protocol	
		Bit #0 ... Bit #15	Bit #4 ... Bit #19	Unit with 16-bit protocol: In 0.1 °C. Value range with 16-bit protocol: [0...65,535]. Unit with 20-bit protocol: In 0.1 °C. Value range with 20-bit protocol: [0...65,535].
		–	Bit #0 ... Bit #3	= 0.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.3 en-US

Code _{HIGH} (hex)	Code _{LOW} (hex)	Data type transmitted by the scan system		
05	35	As of scan system firmware \geq 206x. Only intelliWELD: Purge air flow rate.		
		16-bit protocol	20-bit protocol	
		Bit #0 ... Bit #15	Bit #4 ... Bit #19	Unit with 16-bit protocol: [(SpülLuftdurchflussrate – 4 l/min) * 109,89 min/l]. Value range with 16-bit protocol: [0...2,250]. Unit with 20-bit protocol: [SpülLuftdurchflussrate – 4 l/min] * 109,89 min/l]. Value range with 20-bit protocol: [0...2,250].
		–	Bit #0 ... Bit #3	= 0.
05	36	As of scan system firmware \geq 206x. Only intelliWELD: (Temperature of mirror 2 + 26.6°C).		
		16-bit protocol	20-bit protocol	
		Bit #0 ... Bit #15	Bit #4 ... Bit #19	Unit with 16-bit protocol: In 0.05 °C. Value range with 16-bit protocol: [0...1,992]. Unit with 20-bit protocol: In 0.05 °C. Value range with 20-bit protocol: [0...1,992].
		–	Bit #0 ... Bit #3	= 0.

Notice! Generic information. May be incomplete or even incorrect for your scan system.

Always consult the dedicated scan system manual!

Module Rev.1.0.3 en-US

Code _{HIGH} (hex)	Code _{LOW} (hex)	Data type transmitted by the scan system		
05	37	As of scan system firmware \geq 206x. Only intelliWELD: (Temperature of mirror 1 + 26.6°C).		
		16-bit protocol	20-bit protocol	
		Bit #0 ... Bit #15	Bit #4 ... Bit #19	Unit with 16-bit protocol: In 0.05 °C. Value range with 16-bit protocol: [0...1,992]. Unit with 20-bit protocol: In 0.05 °C. Value range with 20-bit protocol: [0...1,992].
		–	Bit #0 ... Bit #3	= 0.
05	38	As of scan system firmware \geq 206x. Only intelliWELD: Protective-window temperature.		
		16-bit protocol	20-bit protocol	
		Bit #0 ... Bit #15	Bit #4 ... Bit #19	Unit with 16-bit protocol: In 0,044 °C. Value range with 16-bit protocol: [0...2,250]. Unit with 20-bit protocol: In 0,044 °C. Value range with 20-bit protocol: [0...2,250].
		–	Bit #0 ... Bit #3	= 0.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.3 en-US

Code _{HIGH} (hex)	Code _{LOW} (hex)	Data type transmitted by the scan system		
05	39	As of scan system firmware \geq 206x. Only intelliWELD: Collimator temperature.		
		16-bit protocol	20-bit protocol	
		Bit #0 ... Bit #15	Bit #4 ... Bit #19	Unit with 16-bit protocol: In 0.05 °C. Value range with 16-bit protocol: [0...2,250]. Unit with 20-bit protocol: In 0.05 °C. Value range with 20-bit protocol: [0...2,250].
		–	Bit #0 ... Bit #3	= 0.
05	3A	As of scan system firmware \geq 206x. Only intelliWELD: Galvanometer scanner mount temperature.		
		16-bit protocol	20-bit protocol	
		Bit #0 ... Bit #15	Bit #4 ... Bit #19	Unit with 16-bit protocol: In 0.05 °C. Value range with 16-bit protocol: [0...2,250]. Unit with 20-bit protocol: In 0.05 °C. Value range with 20-bit protocol: [0...2,250].
		–	Bit #0 ... Bit #3	= 0.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!

Module Rev.1.0.3 en-US

Code _{HIGH} (hex)	Code _{LOW} (hex)	Data type transmitted by the scan system		
05	3B	As of scan system firmware \geq 206x. Only intelliWELD: [Coolant-flow rate + 0.93 $\text{I} \times \text{min}^{-1}$].		
		16-bit protocol	20-bit protocol	
		Bit #0 ... Bit #15	Bit #4 ... Bit #19	Unit with 16-bit protocol: In 0,00525 $\text{I} \times \text{min-1}$. Value range with 16-bit protocol: [0...2,250]. Unit with 20-bit protocol: [0...2,250]. Value range with 20-bit protocol: In 0,00525 $\text{I} \times \text{min-1}$.
		–	Bit #0 ... Bit #3	= 0.
		As of scan system firmware \geq 206x. Only intelliWELD: Protective-window scattered light value.		
05	3C	16-bit protocol	20-bit protocol	
		Bit #0 ... Bit #15	Bit #4 ... Bit #19	Unit with 16-bit protocol: In 0.00444 V. Value range with 16-bit protocol: [0...2,250]. Unit with 20-bit protocol: [0...2,250]. Value range with 20-bit protocol: In 0.00444 V.
		–	Bit #0 ... Bit #3	= 0.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.3 en-US

Code _{HIGH} (hex)	Code _{LOW} (hex)	Data type transmitted by the scan system		
05	3D	As of scan system firmware ≥ 206x. Only intelliWELD: Flags sensor board Value range with 16-bit protocol: [0...65,535]. Value range with 20-bit protocol: [0...65,535].		
		16-bit protocol	20-bit protocol	
		Bit #15 (MSB)	Bit #19 (MSB)	= 1: Protective-window temperature value not in [0...1,882].
		Bit #14	Bit #18	= 1: Temperature of mirror 1 value not in [0...2,250].
		Bit #13	Bit #17	= 1: Temperature of mirror 2 value not in [0...2,250].
		Bit #10	Bit #14	= 1: Emerging-beam-opening temperature value not in [0...1,200]. Additionally, an INTERLOCK error is initiated.
		Bit #3	Bit #7	= 1: Protective-window scattered light value not in [0...2,250].
		Bit #2	Bit #6	= 1: Coolant-flow rate value not in [750...2,250].
		Bit #1	Bit #5	= 1: Galvanometer-mount temperature value not in [0...1,600]. Additionally, an INTERLOCK error is initiated.
		Bit #0	Bit #4	= 1: Collimator temperature value not in [0...2,000]. Additionally, an INTERLOCK error is initiated.
		–	Bit #0	= 0.
		
			Bit #3	= 0.
05	3E	Reserved.		
		16-bit protocol	20-bit protocol	
		Bit #0	Bit #0	Reserved.
		
		Bit #15	Bit #19	

Notice! Generic information. May be incomplete or even incorrect for your scan system.

Always consult the dedicated scan system manual!

Module Rev.1.0.3 en-US

Code _{HIGH} (hex)	Code _{LOW} (hex)	Data type transmitted by the scan system		
05	3F	As of scan system firmware \geq 2072. Set scaling factor for position values.		
		16-bit protocol	20-bit protocol	
		Bit #2	Bit #6	Reserved.
		
		Bit #15	Bit #19	
		Bit #0	Bit #4	Scaling factor = $1/2^n$. 16-bit protocol $n = 2 \times \text{Bit } \#1 + \text{Bit } \#0$. 20-bit protocol $n = 2 \times \text{Bit } \#5 + \text{Bit } \#4$.
		...	Bit #5	
		Bit #1	Bit #0	= 0.
		–	...	
		Bit #3		
05	49	Reserved.		
05	5E	Reserved.		
05	64	As of scan system firmware \geq 5100. Refer to "excelliSCAN scan heads – Functional Principle of SCANahead Servo Control and Operation by RTC6 Boards" Manual.		
05	65	As of scan system firmware \geq 5100. Refer to "excelliSCAN scan heads – Functional Principle of SCANahead Servo Control and Operation by RTC6 Boards" Manual.		
05	74	Reserved.		
0A		As of scan system firmware \geq 2078. UpdatePermanentMemory UpdatePermanentMemory causes the scan system to set the current servo behavior as the startup behavior for subsequent restarts or resets. See also Chapter 8.1.8 "Configuring the Start Behavior", page 222 .		
		As of scan system firmware \geq 5050. Only excelliSCAN: excelliSCAN scan heads do not support UpdatePermanentMemory .		
		Code _{LOW} (hex)		
		00	Only allowed value.	

Notice! Generic information. May be incomplete or even incorrect for your scan system.

Always consult the dedicated scan system manual!

Module Rev.1.0.3 en-US

Code _{HIGH} (hex)			
OE	As of scan system firmware \geq 206x. SetControlDefinitionMode SetControlDefinitionMode causes the scan system to transmit information about a specific tuning (or the corresponding control algorithm) over the selected status channel. See also "On SetMode, SetControlDefinitionMode and SetEchoMode", Seite 946.		
Code _{LOW} (hex)	Data type transmitted by the scan system		
00	As of scan system firmware \geq 206x.		
01	Tuning number.		
02	16-bit protocol	20-bit protocol	
03	Bit #15 (MSB)	Bit #19 (MSB)	= 0: Tuning available. = 1: Tuning not available.
	Bit #4	Bit #8	Reserved.
	
	Bit #14	Bit #18	
	Bit #0	Bit #4	Tuning type
	= 0: Vector tuning . = 1: Jump tuning . = 2: Reserved. = 3: Reserved. = 4: As of scan system firmware \geq 5050. SCANahead servo control.
	–	Bit #0	= 0.
		...	
		Bit #3	

Notice! Generic information. May be incomplete or even incorrect for your scan system.

Always consult the dedicated scan system manual!

Module Rev.1.0.3 en-US

Code _{HIGH} (hex)													
11	<p>As of scan system firmware ≥ 2078.</p> <p>SelectControlDefinition</p> <p>Only scan systems equipped with several tunings (or corresponding servo algorithms). SelectControlDefinition causes the scan systems to switch to a certain tuning. See also Chapter 8.1.4 "Selecting the Tuning (Dynamics Setting)", page 215.</p> <p>Code_{LOW} = $00\ldots03$ specifies the tuning number.</p> <p>As of scan system firmware ≥ 5050.</p> <p>Only excelliSCAN:</p> <p>excelliSCAN scan heads do not support SelectControlDefinition. With excelliSCAN scan heads, there is only one tuning.</p> <table border="1"> <thead> <tr> <th>Code_{LOW} (hex)</th><th></th></tr> </thead> <tbody> <tr> <td>00</td><td>Tuning 0. Default setting.</td></tr> <tr> <td>01</td><td>Tuning 1.</td></tr> <tr> <td>02</td><td>Tuning 2.</td></tr> <tr> <td>03</td><td>Tuning 3.</td></tr> </tbody> </table>	Code _{LOW} (hex)		00	Tuning 0. Default setting.	01	Tuning 1.	02	Tuning 2.	03	Tuning 3.		
Code _{LOW} (hex)													
00	Tuning 0. Default setting.												
01	Tuning 1.												
02	Tuning 2.												
03	Tuning 3.												
12	<p>As of scan system firmware ≥ 2072.</p> <p>SetPositionScale</p> <p>SetPositionScale causes the scan system to switch to a certain scaling factor. The scan system servo electronics multiplies the position values received from the RTC-control board by this scaling factor.</p> <p>See also Chapter 8.1.7 "Configuring the Effective Calibration", page 221.</p> <p>Important: The scaling factor needs to be changed in 2 steps: 1. Execute SetPositionScale with Code_{LOW} = 83_{H}. 2. Execute SetPositionScale with Code_{LOW} = $[00_{\text{H}}\ldots03_{\text{H}}]$ (= the desired scaling factor). See also "On SetPositionScale", Seite 947.</p> <p>As of scan system firmware ≥ 5050.</p> <p>Only SCANAhead systems:</p> <p>SCANAhead systems do not support SetPositionScale. With SCANAhead systems, the effective calibration cannot be changed.</p> <table border="1"> <thead> <tr> <th>Code_{LOW} (hex)</th><th></th></tr> </thead> <tbody> <tr> <td>00</td><td>The scaling factor is set to the value 1/1. No scaling. Default setting.</td></tr> <tr> <td>01</td><td>The scaling factor is set to the value 1/2.</td></tr> <tr> <td>02</td><td>The scaling factor is set to the value 1/4.</td></tr> <tr> <td>03</td><td>The scaling factor is set to the value 1/8.</td></tr> <tr> <td>83</td><td>Make ready for scaling factor change.</td></tr> </tbody> </table>	Code _{LOW} (hex)		00	The scaling factor is set to the value 1/1. No scaling. Default setting.	01	The scaling factor is set to the value 1/2.	02	The scaling factor is set to the value 1/4.	03	The scaling factor is set to the value 1/8.	83	Make ready for scaling factor change.
Code _{LOW} (hex)													
00	The scaling factor is set to the value 1/1. No scaling. Default setting.												
01	The scaling factor is set to the value 1/2.												
02	The scaling factor is set to the value 1/4.												
03	The scaling factor is set to the value 1/8.												
83	Make ready for scaling factor change.												

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!

Module Rev.1.0.3 en-US

Code _{HIGH} (hex)		
15	As of scan system firmware \geq 2066. SetPosAcknowledgeLevel SetPosAcknowledgeLevel causes the scan system to switch to a certain PosAck limit value. See also Chapter 8.1.6 "Configuring the PosAck Limit Value", page 221 . See also "On SetPosAcknowledgeLevel", Seite 947 .	
	Code _{LOW} (hex)	
	00	Desired PosAck limit value in LSB16 , 16-bit protocol. In bits.
	...	
	FF	
17	As of scan system firmware \geq 2078. Store/RestoreTransmissionMode Store/RestoreTransmissionMode causes with Code_{LOW} = FF_H the scan system to cache the data type currently selected for transmission. It can be reinstated at a later time then with Code_{LOW} = 00_H . See also Data = 0527_H . See also "On Store/RestoreTransmissionMode", Seite 947 .	
	Code _{LOW} (hex)	
	FF	Store temporarily
	00	Reinstate

Notice! Generic information. May be incomplete or even incorrect for your scan system.

Always consult the dedicated scan system manual!

Module Rev.1.0.3 en-US

Code _{HIGH} (hex)	
21	<p>As of scan system firmware \geq 2078.</p> <p>SetEchoMode</p> <p>SetEchoMode verifies whether data transfer is intact.</p> <p>See also Chapter 8.1.9 "Fault Diagnosis and Functional Test", page 222.</p> <p>SetEchoMode passes an 8-bit value to the scan system with Code_{LOW}. As a consequence, the scan system transmits a 16-bit value (16-bit protocol) or 20-bit value (20-bit protocol) on the corresponding status channel to the RTC-control board. If the data transmission has been error-free, with this:</p> <ul style="list-style-type: none"> • The upper 8 bits and Code_{LOW} are identical • With the 16-bit value, the lower 8 bits / with the 20-bit value, the next lower 8 bits and the complementary value of Code_{LOW} (NOT Code_{LOW}) are identical <p>Example with 16-bit protocol: For <code>control_command(1, 1, 0x210A)</code> and if data transfer is error-free, <code>(get_value(1) AND 0xFFFF)</code> returns <code>0xAF5</code>.</p> <p>Example with 20-bit protocol: For <code>control_command(1, 1, 0x210A)</code> and if data transfer is error-free, <code>(get_value(1) AND 0xFFFF0)</code> returns <code>0xAF50</code>.</p> <p>See also "On SetMode, SetControlDefinitionMode and SetEchoMode", Seite 946.</p>
40	<p>As of scan system firmware \geq 5102 + \geq 5112.</p> <p>Refer to "excelliSCAN scan heads – Functional Principle of SCANahead Servo Control and Operation by RTC6 Boards" Manual.</p>
50	<p>As of scan system firmware \geq 5100.</p> <p>Refer to "excelliSCAN scan heads – Functional Principle of SCANahead Servo Control and Operation by RTC6 Boards" Manual.</p>
51	<p>As of scan system firmware \geq 5100.</p> <p>Refer to "excelliSCAN scan heads – Functional Principle of SCANahead Servo Control and Operation by RTC6 Boards" Manual.</p>

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.3 en-US

Comments (RTC6)

- The scan system firmware a.bb.c is read as abbc, for example, 5100 corresponds to version 5.10.0, and 5054 to version 5.05.4.
- On **SetMode**, **SetControlDefinitionMode** and **SetEchoMode**
 - The data type selected by `control_command(CodeHIGH = 05H, 0EH or 21H or Data = 1700H)` is transmitted until another data type is selected.
 - Data transmitted from the scan system to the RTC control board can be queried by **get_value**, **get_values**, **set_trigger/set_trigger4** and **get_waveform**. The first data after switching the data source are transmitted only after a short time delay due to the serial transmission times. Therefore, a delay time of up to 60 s may occur between the switchover time and the first output of the new data type. **control_command** always automatically inserts a waiting time of 60 μ s after the data source is switched (the previously mentioned commands always return correct values).
 - All data returned from the scan system to the RTC6 are passed over as signed 20-bit values. This applies even if the **RTC6 DLL** is set to **RTC4 Compatibility Mode** and even for scan systems without SL2-100 interface, controlled by an **XY2-100 Converter (Accessory)**. Queried data returned by **get_value**, **get_values** or **get_waveform** are nevertheless generally transferred to the PC as signed 32-bit 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(CodeHIGH = 05H, 0EH or 21H or Data = 1700H)`. It is returned in addition to the selected data. In contrast, if **iDRIVE** scan systems (*without* an integrated SL2-100 interface) are controlled by an **XY2-100 Converter (Accessory)**, **get_head_status** returns the
- **XY2-100 status word** only if this signal has been previously selected to be returned from the scan system by **control_command**, see also **Section "Reading Out Data", page 212**.
 - 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.
- On **SetMode**
 - Set position (angular position) values returned by the scan system correspond to the effective output values `Sample<AX...BY>_Out` with **set_trigger/set_trigger4**. Additionally, the 20-bit set position values returned from a z axis in **RTC5 Compatibility Mode** (and in **RTC4 Compatibility Mode** from the x axis and y axis, too) are scaled by a factor 16 relating to the therefore specified 16-bit coordinate values.
 - The angle bit-values (actual position, set position, position error) or angle bit/ms-values (actual speed) returned to the RTC6 can be converted into $^{\circ}$ -values or $^{\circ}/\text{ms}$ -values by the scan system's calibration angle. The calibration angle can be read out by `Data = 0523H`.
 - 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. Therefore, observe the manual of the respective scan system. This manual lists the command codes for currently available sensors.



- On **SetPositionScale**
 - If the scaling factor for scaling the position values is changed, then the user program calibration factor for calculating RTC6 control values (in bits) from the desired **Image field** position values (in mm), see [Chapter 7.3.2 "Image Field Size and Image Field Calibration"](#), [page 166](#), needs to be subjected to an inverse proportional scaling. In addition, the jump speed and mark speed should be adjusted.
 - The currently set scaling factor can be queried with `Data = 053FH`.
- On **SetPosAcknowledgeLevel**
 - The limit value must be specified related to a 16-bit position range.
 - The default start behavior is for the scan system to set the limit value to 0.28% of the *full* angular range of an axis (LSB16 or 1/65536, `CodeLOW = 183 = B7H`) after every power-up or reset.
If other limit values are desired, they must be separately set for each axis (`CodeHIGH = 15H`). SCANLAB recommends setting only limit values (`CodeLOW`) > 14_H (that is, 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).
- On **Store/RestoreTransmissionMode**
 - `control_command(Data = 1700H)` has no effect if a power-up or reset was executed after the most recent execution of `control_command(Data = 17FFH)`.

21 Appendix G: Change Index

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

In this Chapter:

- Changes to Document Revision 1.0.0 en-US from Document Revision 0.0, Seite 949
- Changes to Document Revision 1.0.1 en-US from Document Revision 1.0.0 en-US, Seite 949
- Changes to Document Revision 1.0.2 en-US from Document Revision 1.0.1 en-US, Seite 950
- Changes to Document Revision 1.0.3 en-US from Document Revision 1.0.2 en-US, Seite 951
- Changes to Document Revision 1.0.4 en-US from Document Revision 1.0.3 en-US, Seite 953
- Changes to Document Revision 1.0.5 en-US from Document Revision 1.0.4 en-US, Seite 954
- Changes to Document Revision 1.0.6 en-US from Document Revision 1.0.5 en-US, Seite 955
- Changes to Document Revision 1.0.7 en-US from Document Revision 1.0.6 en-US, Seite 955
- Changes to Document Revision 1.0.8 en-US from Document Revision 1.0.7 en-US, Seite 956
- Changes to Document Revision 1.0.9 en-US from Document Revision 1.0.8 en-US, Seite 956
- Changes to Document Revision 1.0.10 en-US from Document Revision 1.0.9 en-US, Seite 957
- Changes to Document Revision 1.0.11 en-US from Document Revision 1.0.10 en-US, Seite 957
- Changes to Document Revision 1.0.12 en-US from Document Revision 1.0.11 en-US, Seite 958
- Changes to Document Revision 1.0.13 en-US from Document Revision 1.0.12 en-US, Seite 958
- Changes to Document Revision 1.0.14 en-US from Document Revision 1.0.13 en-US, Seite 959
- Changes to Document Revision 1.0.15 en-US from Document Revision 1.0.14 en-US, Seite 960
- Changes to Document Revision 1.0.16 en-US from Document Revision 1.0.15 en-US, Seite 961
- Changes to Document Revision 1.0.17 en-US from Document Revision 1.0.16 en-US, Seite 962
- Changes to Document Revision 1.0.18 en-US from Document Revision 1.0.17 en-US, Seite 963
- Changes to Document Revision 1.0.19 en-US from Document Revision 1.0.18 en-US, Seite 964
- Changes to Document Revision 1.0.20 en-US from Document Revision 1.0.19 en-US, Seite 965



Changes to Document Revision 1.0.0 en-US from Document Revision 0.0

Where	What
Global	Initial Document-Revision. Document Revision <ul style="list-style-type: none"> • 1.0.0 en-US applies to RTC6 Software Package <ul style="list-style-type: none"> • V1.5.0^(a)
Appendix G: Change Index, page 948	

(a) See also [RTC6_Software_RevisionHistory_<Date>_<Rev>.pdf](#).

Changes to Document Revision 1.0.1 en-US from Document Revision 1.0.0 en-US

Where	What
Global	Document Revision <ul style="list-style-type: none"> • 1.0.1 en-US applies to RTC6 Software Package <ul style="list-style-type: none"> • V1.5.2^(a)
eth_get_com_timeouts , page 362	Software change. ^(a) RTC6 DLL-only settings are now possible even without access to an RTC6 Ethernet Board.
eth_set_com_timeouts , page 375	Same as eth_get_com_timeouts .
load_disk , page 489	Software change. ^(a) Has now a version control.
range_checking , page 566	Software change. ^(a) Mode = 2 added: a simulate_ext_stop is passed on to all slave boards.
save_disk , page 591	Same as load_disk .
set_wobbel_vector , page 758	Editorial enhancement. To activate "Freely definable wobbel shape" by set_wobbel_control and set_wobbel_mode (Mode = 3), see page 760 .
sync_slaves , page 789	Software change. ^(a) Has no function anymore.
Appendix G: Change Index, page 948	

(a) See also [RTC6_Software_RevisionHistory_<Date>_<Rev>.pdf](#).



Changes to Document Revision 1.0.2 en-US from Document Revision 1.0.1 en-US

Where	What
Global	Document Revision <ul style="list-style-type: none"> • 1.0.2 en-US applies to RTC6 Software Package <ul style="list-style-type: none"> • V1.6.0^(a)
Section "Folder iSCANCfg", page 31	Software change. ^(a) iSCANCfg6.exe is replaced in the RTC6 Software Package by the generic iSCANCfg.exe.
Chapter 7.4.1 "Enabling, Activating and Switching Laser Control Signals"	Editorial change. Figure 51, page 184.
control_command, page 345	Editorial change. Description of "PowerOK" and "TempOK" has been changed, see XY2-100 status word. Furthermore, the description of 0528 _H and 0529 _H has been removed from the document (these data signal types are not intended for users).
get_error, page 395	Editorial change. error bit Bit #15 no longer "reserved", see page 397.
Chapter 16.2.13 "Master Socket Connector, Slave Socket Connector"	Editorial change. New safety notice on Master/Slave-connected RTC6 Ethernet Boards, see page 875.
Appendix G: Change Index, page 948	

(a) See also [RTC6_Software_RevisionHistory_<Date>_<Rev>.pdf](#).

Changes to Document Revision 1.0.3 en-US from Document Revision 1.0.2 en-US

Where	What
Global	Document Revision <ul style="list-style-type: none"> • 1.0.3 en-US applies to RTC6 Software Package <ul style="list-style-type: none"> • V1.6.1^(a)
Global	Editorial change. The previously used term “Online Positioning” (due to the introduction of “Global Online Positioning”) now reads - where applicable - “Local Online Positioning”, for example, in Chapter 8.3.1 ““Local Online Positioning””, page 227.
Chapter 8.3.2 ““Global Online Positioning””, page 230	Editorial enhancement. Description of “Global Online Positioning”.
Chapter 8.4 “Wobbel Mode”	Editorial enhancement. Section “Example Code (C++)”, page 232.
Chapter 8.6.12 ““Fly Extension” Commands”, page 258	Editorial enhancement. Description of “Fly Extension” Commands.
activate_fly_1_axis , page 317	Software change. ^(a) New command.
activate_fly_2_axes , page 318	Software change. ^(a) New command.
control_command , page 345	Editorial change. The description of 052B _H and 052C _H has been removed from the document (these data signal types are not intended for users).
fly_return_1_axis , page 385	Software change. ^(a) New command.
fly_return_2_axes , page 386	Software change. ^(a) New command.
fly_return_3_axes , page 387	Software change. ^(a) New command.
get_startstop_info , page 420	Software change. ^(a) New Bit #14.
park_position_1_axis , page 556	Software change. ^(a) New command.
park_position_2_axes , page 557	Software change. ^(a) New command.
park_return_1_axis , page 560	Software change. ^(a) New command.
park_return_2_axes , page 561	Software change. ^(a) New command.
set_fly_1_axis , page 632	Software change. ^(a) New command.
set_fly_2_axes , page 633	Software change. ^(a) New command.
set_fly_3_axes , page 637	Software change. ^(a) New command.
set_mcbsp_global_matrix , page 677	Software change. ^(a) New command.
set_mcbsp_global_matrix_list , page 677	Software change. ^(a) New command.
set_mcbsp_global_rot , page 678	Software change. ^(a) New command.
set_mcbsp_global_rot_list , page 678	Software change. ^(a) New command.
set_mcbsp_global_x , page 679	Software change. ^(a) New command.



Where (cont'd.)	What (cont'd.)
set_mcbsp_global_x_list , page 679	Software change. ^(a) New command.
set_mcbsp_global_y , page 680	Software change. ^(a) New command.
set_mcbsp_global_y_list , page 680	Software change. ^(a) New command.
store_timestamp_counter , page 783	Software change. ^(a) New command.
store_timestamp_counter_list , page 783	Software change. ^(a) New command.
wait_for_1_axis , page 820	Software change. ^(a) New command.
wait_for_2_axes , page 822	Software change. ^(a) New command.
wait_for_timestamp_counter , page 830	Software change. ^(a) New command.
Appendix G: Change Index, page 948	

(a) See also [RTC6_Software_RevisionHistory_<Date>_<Rev>.pdf](#).



Changes to Document Revision 1.0.4 en-US from Document Revision 1.0.3 en-US

Where	What
Global	Document Revision <ul style="list-style-type: none"> • 1.0.4 en-US applies to RTC6 Software Package <ul style="list-style-type: none"> • V1.7.0^(a)
Section "Folder RTC6 Tools", page 33	Software change. ^(a) New file. \geq BIOS-ETH 26 is a prerequisite for using the Standalone Functionality .
<code>eth_boot_dcmd</code> , page 353	Software change. ^(a) New command.
<code>set_eth_boot_control</code> , page 627	Software change. ^(a) New command.
<code>eth_boot_timeout</code> , page 353	Software change. ^(a) New command.
<code>set_free_variable</code> , page 649	Software change. ^(a) Changed command.
<code>read_image_eth</code> , page 572	Software change. ^(a) New command.
<code>store_program</code> , page 782	Software change. ^(a) New command.
<code>write_image_eth</code> , page 841	Software change. ^(a) New command.
Chapter 16.2.15 "Real-Time Clock", page 879	Editorial enhancement. Description of the RTC6 Ethernet Board real-time clock.
Chapter 16.7 "Standalone Functionality", page 890	Software change. ^(a) New functionality for RTC6 Ethernet Boards: Standalone Functionality .
Chapter 18 "Appendix C: The RTC6 EtherBox", page 908	Editorial enhancement. Description of the RTC6 EtherBox.
Appendix G: Change Index, page 948	

(a) See also [RTC6_Software_RevisionHistory_<Date>_<Rev>.pdf](#).



Changes to Document Revision 1.0.5 en-US from Document Revision 1.0.4 en-US

Where	What
Global	Document Revision <ul style="list-style-type: none"> • 1.0.5 en-US applies to RTC6 Software Package <ul style="list-style-type: none"> • V1.7.5^(a)
Chapter 4.6.1 "LASER Connector", page 72	Editorial enhancement. Note for laserDESK users on external documents.
<code>eth_boot_timeout</code> , page 353	Editorial change. Command name is <code>eth_boot_timeout</code> (not: <code>set_eth_boot_timeout</code>).
<code>eth_get_com_timeouts_auto</code> , page 363	Software change. ^(a) New command.
<code>eth_set_com_timeouts</code> , page 375	Software change. ^(a) Changed command.
<code>eth_set_com_timeouts_auto</code> , page 376	Software change. ^(a) New command.
<code>get_head_status</code> , page 402	Editorial change. Description corrected.
<code>para_mark_abs</code> , page 549	Software change. ^(a) Changed command.
<code>wait_for_timestamp_counter_mode</code> , page 832	Software change. ^(a) New command.
Chapter 16.2.4 "LASER Socket Connector", page 864	Editorial enhancement. Note for laserDESK users on external documents.
Appendix G: Change Index, page 948	

(a) See also [RTC6_Software_RevisionHistory_<Date>_<Rev>.pdf](#).



Changes to Document Revision 1.0.6 en-US from Document Revision 1.0.5 en-US

Where	What
Global	Document Revision <ul style="list-style-type: none"> • 1.0.6 en-US applies to RTC6 Software Package <ul style="list-style-type: none"> • V1.7.6^(a)
Section "Folder RTC6 Files", page 32	Software change. ^(a) Other internal structure of the RTC6 Software Package as well as more files.
<code>get_startstop_info</code> , page 420	Software change. ^(a) New Bit #15.
<code>set_laser_delays</code> , page 662	Editorial change. Default setting after <code>load_program_file</code> corrected.
<code>wait_for_encoder_mode</code> , page 827	Editorial change. Allowed value range of <code>Value</code> corrected.
<code>wait_for_encoder_in_range_mode</code> , page 826	Editorial change. Allowed value range of <code>EncXmin</code> corrected.
<code>eth_configure_link_loss</code> , page 355	Software change. ^(a) New command.
Chapter 16.5.4 "Checking the Connection to the RTC6 Ethernet Board", page 888	Editorial enhancement.
Chapter 16.9.3 "TI-RTOS", page 899	Editorial enhancement. The RTC6 Ethernet Board uses 3rd party software.
Appendix G: Change Index, page 948	

(a) See also [RTC6_Software_RevisionHistory_<Date>_<Rev>.pdf](#).

Changes to Document Revision 1.0.7 en-US from Document Revision 1.0.6 en-US

Where	What
Global	Document Revision <ul style="list-style-type: none"> • 1.0.7 en-US applies to RTC6 Software Package <ul style="list-style-type: none"> • V1.7.7^(a)
<code>load_program_file</code> , page 501	Editorial enhancement.
Chapter 18 "Appendix C: The RTC6 EtherBox", page 908	Hardware change. Figure 92 , Figure 93 .
Appendix G: Change Index, page 948	

(a) See also [RTC6_Software_RevisionHistory_<Date>_<Rev>.pdf](#).



Changes to Document Revision 1.0.8 en-US from Document Revision 1.0.7 en-US

Where	What
Global	Document Revision <ul style="list-style-type: none"> • 1.0.8 en-US applies to RTC6 Software Package <ul style="list-style-type: none"> • V1.7.8^(a)
Chapter 8.12 "Time Measurements", page 280	Editorial enhancement. 64-bit "Timestamp Counter".
get_timestamp_long, page 430	Software change. ^(a) New command.
wait_for_timestamp_counter_long, page 831	Software change. ^(a) New command.
Appendix G: Change Index, page 948	

(a) See also [RTC6_Software_RevisionHistory_<Date>_<Rev>.pdf](#).

Changes to Document Revision 1.0.9 en-US from Document Revision 1.0.8 en-US

Where	What
Global	Document Revision <ul style="list-style-type: none"> • 1.0.9 en-US applies to RTC6 Software Package <ul style="list-style-type: none"> • V1.7.9^(a)
Chapter 19 "Appendix D: RTC6 Software Packages – History", page 915	Editorial enhancement.
Appendix G: Change Index, page 948	

(a) See also [RTC6_Software_RevisionHistory_<Date>_<Rev>.pdf](#).



Changes to Document Revision 1.0.10 en-US from Document Revision 1.0.9 en-US

Where	What
Global	Document Revision <ul style="list-style-type: none"> • 1.0.10 en-US applies to RTC6 Software Package <ul style="list-style-type: none"> • V1.7.11^(a)
Chapter 7.4.11 "Pulse Synchronization Mode", page 207	Software change. ^(a)
Chapter 8.2 "Coordinate Transformations", page 223	Editorial enhancement. Notes on data recording, page 225 .
<code>eth_get_last_error</code> , page 366	Software change. ^(a) New Bit 27.
<code>regulation3</code> , page 581	Software change. ^(a) New command.
<code>set_defocus_list</code> , page 616	Editorial enhancement. New comment, page 616 .
<code>set_duty_cycle_table</code> , page 622	Software change. ^(a) New command.
<code>set_laser_pulse_sync</code> , page 666	Software change. ^(a) New command.
<code>set_laser_timing_table</code> , page 670	Software change. ^(a) New command.
Appendix G: Change Index, page 948	

(a) See also [RTC6_Software_RevisionHistory_<Date>_<Rev>.pdf](#).

Changes to Document Revision 1.0.11 en-US from Document Revision 1.0.10 en-US

Where	What
Global	Document Revision <ul style="list-style-type: none"> • 1.0.11 en-US applies to RTC6 Software Package <ul style="list-style-type: none"> • V1.7.11^(a)
Chapter 18 "Appendix C: The RTC6 EtherBox", page 908	Product renamed from "RTC6 Ethernet Plug-in+Box" to "RTC6 EtherBox". Removed: #145963, #146152, #146155, #146368, #146371. Added: #148520. #148519.
Appendix G: Change Index, page 948	

(a) See also [RTC6_Software_RevisionHistory_<Date>_<Rev>.pdf](#).



Changes to Document Revision 1.0.12 en-US from Document Revision 1.0.11 en-US

Where	What
Global	Document Revision <ul style="list-style-type: none"> • 1.0.12 en-US applies to RTC6 Software Package <ul style="list-style-type: none"> • V1.7.12^(a)
Section "Folder RTC6 Tools", page 33	Software change. ^(a) BIOS-ETH 29.
Chapter 16.7.1 "Upgrading BIOS-ETH", page 891	Software change. ^(a) BIOS-ETH 29.
Appendix G: Change Index, page 948	

(a) See also [RTC6_Software_RevisionHistory_<Date>_<Rev>.pdf](#).

Changes to Document Revision 1.0.13 en-US from Document Revision 1.0.12 en-US

Where	What
Global	Document Revision <ul style="list-style-type: none"> • 1.0.13 en-US applies to RTC6 Software Package <ul style="list-style-type: none"> • V1.7.12^(a)
control_command, page 345	Editorial change. 052A _H : Stop Event Code 000D0 _H is not intended for users.
Chapter 14.1 "EU Declaration of Conformity – RTC6 PCIe Board", page 852	Editorial change. Replaces Chapter "Compliance with EC Directive for Electromagnetic Compatibility (EMC)".
Chapter 16.9.1 "EU Declaration of Conformity – RTC6 Ethernet Board", page 898	Editorial change. Replaces Chapter "Compliance with EC Directive for Electromagnetic Compatibility (EMC)".
Chapter 18.4.1 "EU Declaration of Conformity – RTC6 EtherBox", page 913	Editorial enhancement.
Chapter 18.4.2 "Compliance with FCC Rules", page 914	Editorial enhancement.
Appendix G: Change Index, page 948	

(a) See also [RTC6_Software_RevisionHistory_<Date>_<Rev>.pdf](#).



Changes to Document Revision 1.0.14 en-US from Document Revision 1.0.13 en-US

Where	What
Global	Document Revision <ul style="list-style-type: none"> • 1.0.14 en-US applies to RTC6 Software Package <ul style="list-style-type: none"> • V1.9.0^(a)
Section "Subfolder Linux", page 32	Software change. ^(a) New subfolder.
Section "Folder RTC6 Tools", page 33	Software change. ^(a) BIOS-ETH 30.
clear_fly_overflow, page 339	Editorial change. Mode = 63 (not = 15).
clear_fly_overflow_ctrl, page 339	Software change. ^(a) New command.
eth_get_standalone_status, page 369	Software change. ^(a) New command.
get_rtc_version, page 417	Software change. ^(a) Bit #11.
set_sky_writing, page 719	Editorial enhancement. RTC5→RTC6.
set_sky_writing_list, page 721	Editorial enhancement. RTC5→RTC6.
set_sky_writing_para, page 724	Editorial enhancement. RTC5→RTC6.
set_sky_writing_para_list, page 726	Editorial enhancement. RTC5→RTC6.
Chapter 16.1.3 "Options", page 860	New Option "LDSA" (RTC6 Ethernet Boards only).
Chapter 16.7.1 "Upgrading BIOS-ETH", page 891	Software change. ^(a) BIOS-ETH 30.
Chapter 16.7.6 "Automatic Booting – Process in Detail", page 896	Editorial enhancement.
Chapter 19 "Appendix D: RTC6 Software Packages – History", page 915	Software change. ^(a) V1.9.0.
Appendix G: Change Index, page 948	

(a) See also [RTC6_Software_RevisionHistory_<Date>_<Rev>.pdf](#).



Changes to Document Revision 1.0.15 en-US from Document Revision 1.0.14 en-US

Where	What
Global	Document Revision <ul style="list-style-type: none"> • 1.0.15 en-US applies to RTC6 Software Package <ul style="list-style-type: none"> • V1.9.0^(a)
Chapter 2.10 "Notes for RTC4 Users", page 47	Editorial enhancement. Encoder counter direction of RTC4 boards, see Notice! , page 56.
control_command , page 345	Editorial change. Content moved to Chapter 20 "Appendix E: iDRIVE Scan Systems – Control Commands and Signals Transmitted to RTC Control Boards" , page 916.
list_repeat , page 479	Editorial enhancement. New comment, page 479.
Chapter 16.3.1 "Hardware Installation", page 880	Editorial enhancement. Figure 88 : voltage supply of multiple RTC6 Ethernet Boards/RTC6 EtherBoxes.
Chapter 18.5 "Technical Specifications – RTC6 EtherBox", page 914	Editorial enhancement.
Chapter 20 "Appendix E: iDRIVE Scan Systems – Control Commands and Signals Transmitted to RTC Control Boards", page 916	Editorial enhancement.
Appendix G: Change Index, page 948	

(a) See also [RTC6_Software_RevisionHistory_<Date>_<Rev>.pdf](#).



Changes to Document Revision 1.0.16 en-US from Document Revision 1.0.15 en-US

Where	What
Global	Document Revision <ul style="list-style-type: none">• 1.0.16 en-US applies to RTC6 Software Package <ul style="list-style-type: none">• V1.11.0^(a)
Chapter 1 "About this Manual", page 24	Editorial change.
Section "Folder RTC6 Tools", page 33	Software change. ^(a) BIOS-ETH 32.
acquire_RTC, page 315	Editorial enhancement. New comment, page 315 .
set_wobbel_control, page 751	Software change. ^(a) Ctrl = 7, Ctrl = 8.
Chapter 16.7.1 "Upgrading BIOS-ETH", page 891	Software change. ^(a) BIOS-ETH 32.
Chapter 19 "Appendix D: RTC6 Software Packages – History", page 915	Software change. ^(a) V1.10.0, V1.11.0.
Chapter 20 "Appendix E: iDRIVE Scan Systems – Control Commands and Signals Transmitted to RTC Control Boards", page 916	Editorial change. Module Rev.1.0.1 en-US.
Appendix G: Change Index, page 948	

(a) See also [RTC6_Software_RevisionHistory_<Date>_<Rev>.pdf](#).



Changes to Document Revision 1.0.17 en-US from Document Revision 1.0.16 en-US

Where	What
Global	Document Revision <ul style="list-style-type: none">• 1.0.17 en-US applies to RTC6 Software Package <ul style="list-style-type: none">• V1.11.0^(a)
Chapter 2.8.6 "Slot Cover with 15-pin D-SUB Connector and 9-pin D-SUB Connector", page 45	Editorial enhancement. #130209.
Chapter 16.3.1 "Hardware Installation", page 880	Editorial change. Figure 88.
Chapter 20 "Appendix E: iDRIVE Scan Systems – Control Commands and Signals Transmitted to RTC Control Boards", page 916	Editorial change. Module Rev.1.0.2 en-US.
Appendix G: Change Index, page 948	

(a) See also [RTC6_Software_RevisionHistory_<Date>_<Rev>.pdf](#).



Changes to Document Revision 1.0.18 en-US from Document Revision 1.0.17 en-US

Where	What
Global	Document Revision <ul style="list-style-type: none"> • 1.0.18 en-US applies to RTC6 Software Package <ul style="list-style-type: none"> • V1.12.0^(a)
Chapter 1 "About this Manual", page 24	Editorial change.
Section "Folder RTC6 Tools", page 33	Software change. ^(a) BIOS-ETH 33.
Section "Checking the z axis Calibration", page 169	Editorial enhancement. Notes, page 170 and RTC6 commands from V1.12.0.
get_temperature, page 428	Software change. ^(a) New command.
load_z_table_20b, page 510	Software change. ^(a) New command.
load_z_table_no_20b, page 512	Software change. ^(a) New command.
read_abc_from_file_20b, page 569	Software change. ^(a) New command.
set_wobbel_control, page 751	Software change. ^(a) With Ctrl = 8, now alternation occurs with every Microstep (no longer with every set_wobbel_vector call).
set_wobbel_vector_2, page 762	Software change. ^(a) New command.
write_abc_to_file_20b, page 835	Software change. ^(a) New command.
Chapter 16.7.1 "Upgrading BIOS-ETH", page 891	Software change. ^(a) BIOS-ETH 33.
Chapter 19 "Appendix D: RTC6 Software Packages – History", page 915	Software change. ^(a) V1.12.0.
Appendix G: Change Index, page 948	

(a) See also RTC6_Software_RevisionHistory_<Date>_<Rev>.pdf.



Changes to Document Revision 1.0.19 en-US from Document Revision 1.0.18 en-US

Where	What
Global	Document Revision <ul style="list-style-type: none"> • 1.0.19 en-US applies to RTC6 Software Package <ul style="list-style-type: none"> • V1.13.0^(a)
Chapter 1 "About this Manual", page 24	Editorial change.
Section "Folder RTC6 Tools", page 33	Software change. ^(a) BIOS-ETH 33.
Chapter 7.3.4 "3D Image Field", page 170	Editorial enhancement. ABC values from the *_ReadMe.txt file of correction file are to be interpreted as 16-bit values.
load_z_table_20b, page 510	Editorial change. Allowed value range for A, B, C.
load_z_table_no_20b, page 512	Editorial change. Allowed value range for A, B, C.
set_mcbsp_out_ptr, page 688	Software change. ^(a) Number Bit 31.
set_mcbsp_out_ptr_list, page 689	Software change. ^(a) New command.
set_wobbel_vector, page 758	Software change. ^(a) Calculation for the variation of the current laser power P after set_wobbel_mode(Mode = 2): formula, page 759 (instead of: $P = P0 \times (1 + dPower \times n)$).
set_wobbel_vector, page 758	Software change. ^(a) Calculation for the variation of the current laser power P after set_wobbel_mode(Mode = 3): formula, page 760 (instead of: $P = P100 \times (factor + dPower \times n)$).
Chapter 16.7.1 "Upgrading BIOS-ETH", page 891	Software change. ^(a) BIOS-ETH 33.
Chapter 19 "Appendix D: RTC6 Software Packages – History", page 915	Software change. ^(a) V1.13.0.
Appendix G: Change Index, page 948	

(a) See also [RTC6_Software_RevisionHistory_<Date>_<Rev>.pdf](#).

Changes to Document Revision **1.0.20 en-US** from Document Revision **1.0.19 en-US**

Where	What
Global	Document Revision <ul style="list-style-type: none"> • 1.0.20 en-US applies to RTC6 Software Package <ul style="list-style-type: none"> • V1.14.1^(a)
Chapter 1 "About this Manual", page 24	Editorial change.
Section "Folder RTC6 Tools", page 33	Software change. ^(a) BIOS-ETH 35.
Chapter 4.4 "Master Socket Connector, Slave Socket Connector", page 65	Hardware change. #149963 (previously: #117241).
<code>eth_set_high_performance_mode</code> , page 377	Software change. ^(a) New command.
<code>get_error</code> , page 395	Software change. ^(a) Error bit Bit #17 no longer "reserved", see page 397 .
<code>get_list_pointer</code> , page 409	Editorial enhancement. New comment, page 409 .
<code>set_laser_control</code> , page 659	Editorial enhancement. Further clarification with Bit #3 and Bit #4.
<code>set_mcbsp_out_oie_ctrl</code> , page 687	Software change. ^(a) New command.
<code>set_mcbsp_out_oie_list</code> , page 687	Software change. ^(a) New command.
<code>set_vector_control</code> , page 745	Software change. Ctrl = 8.
<code>wait_for_encoder_mode</code> , page 827	Editorial change. <code>wait_for_encoder_mode</code> is category "Normal List Command" (not: "Multiple List Command").
<code>write_port_list</code> , page 844	Software change. ^(a) New command.
Chapter 16.7.1 "Upgrading BIOS-ETH", page 891	Software change. ^(a) BIOS-ETH 35.
Chapter 16.8 ""High Performance Mode"", page 897	Software change. ^(a) New functionality for RTC6 Ethernet Boards: "High Performance Mode".



Where (cont'd.)	What (cont'd.)
Chapter 18.2 "Cabling", page 911	Hardware change. #149964 (previously: #148519).
Chapter 18.5 "Technical Specifications – RTC6 EtherBox", page 914	Weight, page 914.
Chapter 19 "Appendix D: RTC6 Software Packages – History", page 915	Software change. ^(a) V1.14.1.
Chapter 20 "Appendix E: iDRIVE Scan Systems – Control Commands and Signals Transmitted to RTC Control Boards", page 916	Editorial change. Module Rev.1.0.3 en-US.
Appendix G: Change Index, page 948	

(a) See also [RTC6_Software_RevisionHistory_<Date>_<Rev>.pdf](#).