MAN05010.250625

R3 programming language

© 2025 Robox SpA

The content of this document is prepared with the utmost care/diligence, and subjected to careful control.
Robox Spa, however, disclaims all liability, direct and indirect, to users and in general to any third party, for any inaccuracies, errors, omissions, damages (direct, indirect, consequential, punishable and sanctionable) resulting from the aforementioned contents

Table of contents

Parte I General index 17

Parte II Constants 17

1 Constants 17

Parte III Functions/instructions 19

1 Functions/instructions index 20
2 Functions error codes 57
3 Return codes of the MV_ and MVA_ functions 61
4 Modbus error codes 62
5 SDO Error Codes (CoE/SoE) 63
6 Zero-cycles types 66
Linear axis: zero cycle towards the micro 66
Linear axis: zero cycle towards the micro + index 69
Linear axis: zero cycle towards the end of stroke 72
Linear axis: zero cycle towards the end of stroke + index 74
One-way rotary axis: zero cycle towards the micro + index with preliminary search for the ..... 76 variation
One-way rotary axis: zero cycle towards the micro + index without preliminary search for t..... 80cro variation
One-way rotary axis: zero cycle towards the micro with preliminary search for the micro va..... 83n
One-way rotary axis: zero cycle towards the micro without preliminary search for the micr..... 85ation
7 Categories 88
Mathematical functions/instructions 88
Safety functions/instructions 90
Rule/task functions/instructions 91
Alarms management 92
Axioline Management 94
Serial channel management 95
CANopen/EtherCAT Management 96
CNET Management 99
BCC Communication Management 99
Flow control management 102
Endianess conversion management 103
Counter management 109
Data Management 110
Flash management 115
I/O forcing management 116
Modbus/TCP client management 116
MOT management 120
Motion management 120
Power set management 123
String management 124
Timer management 125
UDP/IP management 126
Global variable management 127
Miscellaneous 129
8 Functions 133
Function abs() 133

© 2025 Robox SpA

Function abs_mod() 134
Function acos() 134
Function ah_cmd() 135
Function ah_get_d() 136
Function ah_get_s() 137
Function ah_info() 139
Function ah_log() 140
Function ah_log_us() 141
Function al2am() 142
Function alarm_reset() 145
Function alarm_reset_par() 146
Function alarm_reset_par_us() 149
Function alarm_set() 151
Function alarm_set_us() 152
Function as_get_d() 154
Function asin() 155
Function atan() 156
Function atan2() 156
Function axio_iol_r 157
Function axio_iol_w 159
Function axio_pm_r() 160
Function axio_robj() 162
Function axio_wobj() 177
Function b0() 191
Function b1() 192
Function b2() 193
Function b3() 194
Function b4() 194
Function b5() 195
Function b6() 195
Function b7() 196
Function bcc_get_double() 197
Function bcc_get_float() 197
Function bcc_get_i16() 198
Function bcc_get_i32() 198
Function bcc_get_i8() 199
Function bcc_get_string() 199
Function bcc_get_tcp_clients_info() 200
Function bcc_get_u16() 201
Function bcc_get_u32() 201
Function bcc_get_u8() 202
Function bcc_h_setup() 202
Function bcc_receive() 204
Function bcc_reply_ack() 204
Function bcc_reply_msg() 205
Function bcc_reply_nack() 206
Function bcc_send() 206
Function check_mv_follow2() 207
Function check_mva_follow2() 210
Function check_mva_follow2_m() 213
Function co_asw_sdo() 215
Function co_controlword_om() 218
Function co_controlword_state() 220
Function co_get_ws_diagno() 221

© 2025 Robox SpA

Function co_par_download() 224
Function co_par_upload() 227
Function co_rerr() 231
Function co_rnmt() 232
Function co_robj() 234
Function co_rpdo() 241
Function co_send_sdo() 242
Function co_statusword() 246
Function co_sync() 247
Function co_wnmt() 248
Function co_wobj() 249
Function co_wpdo() 254
Function coe_robj() 255
Function coe_wobj() 259
Function command() 264
Function cond_val() 265
Function cos() 265
Function crc16_be() 266
Function crc16_le() 268
Function ct_cv() 269
Function ct_pv() 270
Function ct_qd() 270
Function ct_qu() 271
Function dec() 271
Function delta_enc16() 272
Function delta_enc32() 273
Function diff() 274
Function dw0() 275
Function dw1() 277
Function dwbe2le() 277
Function dwbe2lef() 278
Function dwbe2mbe() 278
Function dwbe2mle() 279
Function dwle2be() 279
Function dwle2bef() 280
Function dwle2mbe() 280
Function dwle2mbef() 281
Function dwle2mle() 281
Function dwle2mlef() 282
Function dwmbe2be() 282
Function dwmbe2le() 283
Function dwmbe2lef() 283
Function dwmle2be() 284
Function ecat_dev_on_off() 284
Function ecat_get_state() 287
Function ecat_probe() 288
Function ecat_set_state() 290
Function excl_lam() 292
Function excl_pam() 293
Function f_avail() 293
Function f_close() 294
Function f_copy() 295
Function f_del() 296
Function f_dir() 297

© 2025 Robox SpA

Function f_format() 298
Function f_gets() 300
Function f_mkdir() 301
Function f_open() 302
Function f_puts() 304
Function f_read() 305
Function f_read_table() 306
Function f_read_table_s() 308
Function f_rename() 310
Function f_seek() 311
Function f_write() 312
Function fall() 314
Function fall_filt() 314
Function filter_lp() 317
Function filter_lp_st() 317
Function filter_notch() 319
Function filter_notch_st() 320
Function first_time() 321
Function force() 322
Function fract() 324
Function get_float() 325
Function get_hw_info() 326
Function get_i16() 328
Function get_i32() 328
Function get_i8() 329
Function get_real() 330
Function get_u16() 331
Function get_u32() 332
Function get_u8() 333
Function get_virt_ow() 334
Function gv_create() 334
Function gv_datatype() 336
Function gv_datatype_byname() 337
Function gv_destroy() 338
Function gv_exist() 338
Function gv_flags() 339
Function gv_flags_byname() 341
Function gv_link() 342
Function gv_lock() 344
Function gv_name() 344
Function gv_read() 345
Function gv_read_partial() 345
Function gv_size() 346
Function gv_size_byname() 347
Function gv_unlink() 347
Function gv_unlock() 348
Function gv_update_file() 348
Function gv_write() 349
Function gv_write_partial() 349
Function in_range() 350
Function inc() 351
Function index_of() 352
Function ipaddr_to_str() 353
Function is_alarm_par_present() 353

© 2025 Robox SpA

Function is_alarm_par_present_us() 356
Function is_alarm_present() 357
Function is_forced() 358
Function kin_conv() 361
Function log() 362
Function log10() 363
Function loop_time() 363
Function master_filter() 364
Function maximum() 364
Function mbc_connect() 365
Function mbc_init() 366
Function mbc_mask_write_reg() 368
Function mbc_mask_write_reg_s() 370
Function mbc_read_coils() 372
Function mbc_read_coils_s() 373
Function mbc_read_discr_inp() 374
Function mbc_read_discr_inp_s() 376
Function mbc_read_hold_reg() 377
Function mbc_read_hold_reg_s() 379
Function mbc_read_inp_reg() 381
Function mbc_read_inp_reg_s() 382
Function mbc_shutdown() 384
Function mbc_write_coils() 384
Function mbc_write_coils_s() 386
Function mbc_write_multiple_coils() 387
Function mbc_write_multiple_coils_s() 388
Function mbc_write_multiple_regs() 390
Function mbc_write_multiple_regs_s() 392
Function mbc_write_reg() 393
Function mbc_write_reg_s() 395
Function mbcu_mask_write_reg() 396
Function mbcu_read_coils() 398
Function mbcu_read_discr_inp() 400
Function mbcu_read_hold_reg() 401
Function mbcu_read_inp_reg() 403
Function mbcu_write_coils() 405
Function mbcu_write_multiple_coils() 406
Function mbcu_write_multiple_regs() 408
Function mbcu_write_reg() 410
Function minimum() 412
Function mod() 412
Function mot_append() 413
Function mot_clear() 417
Function mot_create() 417
Function mot_delete() 421
Function mot_get() 421
Function mot_remove() 423
Function mt_ats() 423
Function mt_ats2() 424
Function mv_cam() 425
Function mv_crimper() 427
Function mv_follow() 428
Function mv_follow2() 432
Function mva_follow2() 442

© 2025 Robox SpA

Function mva_follow2_m() 443
Function mv_mot_exec() 445
Function mv_phase_adj() 447
Function mv_phase_adj2() 448
Function mv_ramp() 451
Function mv_reach_target() 451
Function mv_sine_cam() 453
Function mv_synchro() 454
Function mv_table() 455
Function mv_to() 457
Function mv_to(), parameterized inline 459
Function mv_to_cj() 461
Function mv_to_cjv() 461
Function mv_to_cjv_info() 463
Function mv_to_vel() 464
Function mv_tracking() 466
Function mva_jog() 466
Function mva_jog2() 468
Function mva_open_loop() 470
Function mva_to_n() 471
Function mva_to_n_cj() 475
Function mva_to_n_v() 479
Function mva_zc() 486
Function normalize() 488
Function onoff_filt() 489
Function par_even() 491
Function par_odd() 492
Function planar_comp_cp() 493
Function planar_comp_ip() 494
Function pos_capture_cmd() 495
Function pos_capture_sts() 499
Function pow() 500
Function ps_channel_enable() 501
Function ps_channel_enabled() 504
Function ps_channel_fault() 505
Function ps_channel_reset_fault() 507
Function ps_drive_reset() 509
Function ps_exclude_lax() 511
Function ps_exclude_pax() 513
Function ps_power_enable() 514
Function ps_reinclude_lax() 515
Function ps_reinclude_pax() 516
Function ps_status() 517
Function qwbe2lef() 521
Function qwbedwx2lef() 521
Function qwmbe2lef() 522
Function qwmbedwx2lef() 522
Function r_rol() 523
Function r_ror() 524
Function r_shl() 525
Function r_shr() 526
Function r2m16() 527
Function r2m32() 528
Function r2m8() 529

© 2025 Robox SpA

Function ramp() 530
Function rand32() 531
Function random() 531
Function random_value() 531
Function range() 532
Function range_adj() 533
Function rbit() 534
Function release() 534
Function rise() 536
Function rise_filt() 537
Function round() 539
Function rtc_date() 540
Function rtc_time() 540
Function safe_appl_ack() 541
Function safe_appl_diag() 542
Function safe_diag() 544
Function safe_diag_reset() 546
Function safe_enab_out() 547
Function safe_oper_ack() 548
Function safe_prj_info() 549
Function safe_sts() 550
Function sb0() 551
Function sb1() 552
Function sb2() 553
Function sb3() 553
Function sb4() 554
Function sb5() 554
Function sb6() 555
Function sb7() 556
Function sbit() 556
Function sdw0() 557
Function sdw1() 558
Function ser_485_tx_sts() 559
Function ser_bcc_h_init() 561
Function ser_bin_in() 562
Function ser_bin_out() 563
Function ser_bin_ready() 563
Function ser_buff_free() 564
Function ser_buff_queue() 564
Function ser_flush() 564
Function ser_in() 565
Function ser_mod() 566
Function ser_out() 567
Function ser_r_sts() 567
Function ser_ready() 568
Function set_kinematics() 569
Function set_virt_iw() 569
Function shift_left() 570
Function shift_right() 571
Function shl() 573
Function shr() 573
Function sign() 574
Function similar() 575
Function sin() 576

© 2025 Robox SpA

Function sizeof() 576
Function soe_ridn() 577
Function soe_widn() 589
Function sqrt() 594
Function statistics() 595
Function store_nvr_nopf() 595
Function store_nvrr_nopf() 596
Function str_asc() 598
Function str_chr() 598
Function str_compact() 599
Function str_equal() 600
Function str_find() 600
Function str_format() 601
Function str_i() 602
Function str_insert() 603
Function str_len() 604
Function str_mid() 605
Function str_r() 606
Function str_replace() 607
Function str_to_ipaddr() 608
Function str_val() 608
Function sum8_mem() 611
Function sum16_mem() 611
Function sum32_mem() 612
Function sumr_mem() 612
Function sw0() 612
Function sw1() 613
Function sw2() 614
Function sw3() 614
Function tan() 615
Function time_micro() 616
Function tm_et() 616
Function tm_pt() 617
Function tm_pt2() 618
Function tm_q() 619
Function tobcd() 619
Function tobin() 620
Function todeg() 621
Function tolower() 621
Function torad() 621
Function toupper() 622
Function tracking() 622
Function trunc() 624
Function udp_close() 625
Function udp_open_client() 626
Function udp_open_server() 628
Function udp_recv() 631
Function udp_recv_from() 632
Function udp_recv_notify() 634
Function udp_send() 636
Function udp_send_notify() 638
Function udp_send_to() 640
Function virt_io() 642
Function w0() 644

© 2025 Robox SpA

Function w1() ... 645
Function w2() ... 645
Function w3() ... 646
Function wbe2le() ... 647
Function wbit() ... 647
Function wle2be() ... 648
Function write_dis() ... 648
9 Instructions ... 650
Instruction __main_loop__ ... 650
Instruction _else ... 651
Instruction _if ... 651
Instruction aux ... 652
Instruction axes ... 653
Instruction axes_m ... 654
Instruction b0 ... 654
Instruction b1 ... 654
Instruction b2 ... 655
Instruction b3 ... 655
Instruction b4 ... 655
Instruction b5 ... 655
Instruction b6 ... 656
Instruction b7 ... 656
Instruction bcc_set_double ... 656
Instruction bcc_set_float ... 657
Instruction bcc_set_i16 ... 658
Instruction bcc_set_i32 ... 658
Instruction bcc_set_i8 ... 659
Instruction bcc_set_string ... 659
Instruction bcc_set_u16 ... 660
Instruction bcc_set_u32 ... 660
Instruction bcc_set_u8 ... 661
Instruction bit_mask_add ... 661
Instruction bit_mask_remove ... 663
Instruction bit_mask_set ... 665
Instruction break ... 667
Instruction call ... 667
Instruction case ... 669
Instruction continue ... 670
Instruction ct_cd ... 670
Instruction ct_cdr ... 671
Instruction ct_cu ... 672
Instruction ct_cur ... 673
Instruction ct_cv ... 674
Instruction ct_load ... 674
Instruction ct_pv ... 675
Instruction ct_reset ... 675
Instruction default ... 675
Instruction do ... 676
Instruction dw0 ... 676
Instruction dw1 ... 677
Instruction dwell ... 677
Instruction else ... 677
Instruction end_aux ... 678
Instruction end_do_while ... 679

© 2025 Robox SpA

Instruction end_for 679
Instruction end_fun 680
Instruction end_main_loop 680
Instruction end_motion 681
Instruction end_ref 681
Instruction end_rule 682
Instruction end_select 682
Instruction end_while 683
Instruction endif 683
Instruction excl_lam 684
Instruction excl_pam 685
Instruction fill_mem 685
Instruction for 686
Instruction function 688
Instruction get_i64 692
Instruction get_u64 693
Instruction goto 693
Instruction group 694
Instruction if 696
Instruction inf_report 697
Instruction int_after 698
Instruction int_inp 699
Instruction int_timer 701
Instruction lit 702
Instruction motion 703
Instruction move_byte 704
Instruction move_mem 705
Instruction mt_dis 706
Instruction mt_en 706
Instruction mt_rr_flag 707
Instruction mt_wait 707
Instruction order 708
Instruction out_m 710
Instruction qwbe2le 711
Instruction qwbedwx2le 711
Instruction qwle2be 712
Instruction qwle2bedwx 712
Instruction qwle2bedwxf 713
Instruction qwle2bef 714
Instruction qwle2mbe 714
Instruction qwle2mbedwx 715
Instruction qwle2mbedwxf 715
Instruction qwle2mbef 716
Instruction qwmbe2le 716
Instruction qwmbedwx2le 717
Instruction ref 717
Instruction ref_prologue 718
Instruction restart_point 719
Instruction return 719
Instruction rule 720
Instruction rule_epilogue 728
Instruction rule_freq 729
Instruction rule_init 730
Instruction rule_periodic 730

© 2025 Robox SpA

Instruction rule_prologue 732
Instruction rule_start_norc 732
Instruction sb0 733
Instruction sb1 733
Instruction sb2 733
Instruction sb3 734
Instruction sb4 734
Instruction sb5 734
Instruction sb6 735
Instruction sb7 735
Instruction sdw0 735
Instruction sdw1 736
Instruction select 736
Instruction ser_con 737
Instruction ser_w_sts 738
Instruction set_float 739
Instruction set_i16 740
Instruction set_i32 741
Instruction set_i64 742
Instruction set_i8 743
Instruction set_real 744
Instruction set_u16 745
Instruction set_u32 746
Instruction set_u64 747
Instruction set_u8 747
Instruction set_version 748
Instruction sw0 749
Instruction sw1 749
Instruction sw2 750
Instruction sw3 750
Instruction terminator 750
Instruction tm_et 751
Instruction tm_int 751
Instruction tm_of 753
Instruction tm_on 754
Instruction tm_pt 756
Instruction tm_pt2 756
Instruction tm_reset 756
Instruction tm_sqw 756
Instruction tm_tp 758
Instruction w0 759
Instruction w1 759
Instruction w2 760
Instruction w3 760
Instruction while 760

Parte IV Grammar 761

1 Grammar generalities 761

Parte V Introduction 763

1 R3 language generalities 763

Parte VI Metacommands 770

© 2025 Robox SpA

1 Metacommands 770
2 Metacommand $define 772
3 Metacommand $ifdef/$ifndef 772
4 Metacommand $include 773
5 Metacommand $undef 774

Parte VII Operators 774

1 Operators 774
2 Arithmetical operators 776
3 Binary operators (bit) 777
4 Assignment operators 781
5 String concatenation operators 781
6 Logical operators 782
7 Numerical relational operators 784

Parte VIII Data types 787

1 Structure BCC_MSGUSER 793
2 Structure STRU_AAX 795
3 Structure STRU_AH_D 796
4 Structure STRU_AH_D_EXT 797
5 Structure STRU_AH_INFO 798
6 Structure STRU_AS_D 799
7 Structure STRU_AS_D_EXT 800
8 Structure STRU_AXIO_PDI 801
9 Structure STRU_BCC_TCP_CLIENT 803
10 Structure STRU_CAM 804
11 Structure STRU_CO_OBJ 805
12 Structure STRU_COE_OBJ 807
13 Structure STRU_COERRMSG 809
14 Structure STRU_CRIMPER 812
15 Structure STRU_ECAT_PROBE 814
16 Structure STRU_FILEHND 815
17 Structure STRU_FILEINFO 816
18 Structure STRU_FOLLOW 816
19 Structure STRU_FOLLOW2 819
20 Structure STRU_FOLLOW2_INFO 827
21 Structure STRU_FOLLOW2_S 827
22 Structure STRU_FOLLOW2_SM 834
23 Structure STRU_GROR 841
24 Structure STRU_HW_INFO 841

© 2025 Robox SpA

25 Structure STRU_INT 842
26 Structure STRU_LARGE_INT 844
27 Structure STRU_LARGE_UINT 844
28 Structure STRU_MASTERF 845
29 Structure STRU_MATCAM 846
30 Structure STRU_MODBUS 847
31 Structure STRU_MVGENITP 849
32 Structure STRU_MVITP 850
33 Structure STRU_MVITPCJ 852
34 Structure STRU_MVTO 853
35 Structure STRU_MVTOCJ 855
36 Structure STRU_MVTOCJV 857
37 Structure STRU_MVTOCJV_INFO 859
38 Structure STRU_MVTOCJVE 861
39 Structure STRU_MVZC 863
40 Structure STRU_PHADJ 868
41 Structure STRU_PHADJ2 871
42 Structure STRU_PM_PD 875
43 Structure STRU_REACHTARGET 876
44 Structure STRU_SAFETY_PRJINFO 878
45 Structure STRU_SOE_IDN 879
46 Structure STRU_STAT 882
47 Structure STRU_SYNCHRO 884
48 Structure STRU_TRACK 886
49 Structure STRUCT 888
50 Structure STRUCT_P 889
51 Structure TABSTR 889
52 Type BOOL 890
53 Type CAST 890
54 Type CHAR 891
55 Type COUNTER 892
56 Type FLOAT 892
57 Type I16 893
58 Type I32 894
59 Type I8 895
60 Type POWER_SET 896
61 Type REAL 898
62 Type STRING 899
63 Type TIMER 900

© 2025 Robox SpA

64 Type U16 900
65 Type U32 901
66 Type U8 902

Parte IX Miscellaneous 903

1 LIVE Changes 903

Index 907

© 2025 Robox SpA

General index

Main topics:

- Introduction 763
- Constants 17
- Grammar 761
- Metacommands 770
- Operators 774
- Data types 787
- Functions/Instructions 19

Other useful topics:

- "LIVE" modifications 903

Other useful links:

- RTE Operating System (Real Time Extended)
- Global variables
- Predefined variables
- LADDER IEC1131 Programming Language
- Object Block Library (Interface to OS)
- RPE Library (Robox Path Executor)

Constants

Constants

A constant is a numerical value that does not change during program execution. In writing R3 instructions, the use of constants is allowed.

There are some constants that are specific to the language and cannot be changed by the user.

Predefined mathematical constants

Keyword	Numerical Value	Meaning
K_E	2.718281828459045	Euler constant
K_PI	3.141592653589793	PI
K_PIMEZZI	1.570796326794896	PI half
K_PI_2	1.570796326794896	PI half
K_1_PI	0.318309886183790	One divided PI
K_DUEPI	6.283185307179586	Two PI
K_2PI	6.283185307179586	Two PI
K_EPSILON	1e-6	Decision threshold for comparisons between reals
K_MAXFLOAT	3.37e+38	Largest constant representable in a float
K_MINFLOAT	8.43e-37	Smallest constant representable in a float
K_MAXREAL	1.797693e+308	Largest constant representable in a double

© 2025 Robox SpA

Keyword	Numerical Value	Meaning
K_MINREAL	2.225074e-308	Smallest constant representable in a double

User-definable constants

The types of constants that the programmer can define are:

- integer constants 18
- real constants 18
- string constants 19

Integer constants

They can be any integer between -2147483648 and +2147483647.

Integer constants can be represented in decimal or hexadecimal base.

To identify an hexadecimal number it will be necessary to add the letter "**H**" to the end of the number, or prefix the number with "**0x**".

Hexadecimal numbers must always begin by numeric digit (prefix the 0 if necessary). Hexadecimal numbers are represented in two-complement on 8 digits, and the minus sign is therefore not allowed.

Examples

Example decimal constants:

```
; Decimal constants
10
-20
```

Example hexadecimal constants:

```
0XA ;  Decimal value 10
0AH ; Decimal  value 10
10H ; Decimal  value 16
0FFH ; Decimal  value 255
FFFFFFH ; Decimal  value -1
0XFFFFFF ; Decimal  value -1
```

Real constants

They can be any real number between -1.79 E 308 and 1.79 E 308.

The smallest entity that can be represented is, in absolute value 2.22 E -308.

The significant digits are at least 15.

The syntax of the representation of a real number is as follows: *si.d*

Parameter	Meaning
s	sign of the number (optional)
i	integer part of the number
.	decimal separator (optional)

© 2025 Robox SpA

Parameter	Meaning
d	decimal part of the number (optional)

The syntax of the exponential representation is as follows: *si.dEse*

Parameter	Meaning
s	sign of the number (optional)
i	integer part of mantissa
.	decimal separator (optional)
d	decimal part of the mantissa (optional)
E	exponential format identifier (in base 10)
s	sign of the exponent (optional)
e	exponent

Examples

Example real constants:

```
10
10.51
1452.7
1.4527E3
-12.4567E-4
```

Sting constants

String constants can be any sequence of characters between double quotes " ".

NOTE: To use the double-quote (") character within a string constant the syntax to use is \" . The maximum length is 127 net characters.

Examples

Example usage:

```
; String constant example
"this is a string"
"the doublet character \" is contained in this string "
```

Functions/instructions

List of functions and instructions divided by category:

- Alarms 92
- Axioline 94
- BCC communication 99
- CANopen/EtherCAT 96
- CNET 99

Complete list of all functions and instructions:

- Functions/instructions index 20

Other:

- Functions error codes 57
- Return codes of mv_ and mva_ functions 61

© 2025 Robox SpA

- Counter 109
- Data 110
- Endianess conversion 103
- Flash 115
- Flow control 102
- Global variables 127
- I/O forcing 116
- Mathematical 88
- Modbus/TCP client 116
- MOT 120
- Motion 120
- Power set 123
- Rule/Task 91
- Safety 90
- Serial channel 95
- String 124
- Timer 125
- UDP/IP 126
- Miscellaneous 129
- Modbus error codes 62
- SDO error codes (CoE/SoE) 63
- Zero cycle types 66

Functions/instructions index

List of all functions/instructions available in R3.

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (has a return value)				
Keyword	R	T	F/I	Description
_else 651	R	T	I	Flow control instruction
_if 651	R	T	I	Flow control instruction
abs 133()	R	T	F	Absolute value of a quantity
abs_mod 134()	R	T	F	Remainder of division
acos 134()	R	T	F	Arcocosine
ah_cmd 135()	R	T	F	Commands for alarm history

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (has a return value)				
Keyword	R	T	F/I	Description
ah_get_d 136()	R	T	F	Commands for alarm history
ah_get_s 137()	R	T	F	Commands for alarm history
ah_info 139()	R	T	F	Commands for alarm history
ah_log 140()	R	T	F	Commands for alarm history
ah_log_us 141()	R	T	F	Commands for alarm history
al2am 142()	R	T	F	Returns AM info related to a specific alarm
alarm_reset 145()	R	T	F	Reset an alarm
alarm_reset_par 146()	R	T	F	Reset an alarm by also specifying the parameter
alarm_reset_par_us 149()	R	T	F	Reset an alarm by also specifying the USV parameter
alarm_set 151()	R	T	F	Set an alarm
alarm_set_us 152()	R	T	F	Set an alarm with USV parameter
and 782	R	T	I	AND logic operator
as_get_d 154()	R	T	F	Commands for alarm history
asin 155()	R	T	F	Arcosine
atan 156()	R	T	F	Arcotangent of a value
atan2 156()	R	T	F	Arcotangent of a ratio
aux 652	R	-	I	Start block AUX in the RULE
axes 653	R	-	I	Definition of axes involved in the RULE (constant)
axes_m 654	R	-	I	Definition of axes involved in hte RULE (mask)

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (has a return value)				
Keyword	R	T	F/I	Description
axio_iol_r 157()	-	T	F	Function to read a data area from known master IOlink
axio_iol_w 159()	-	T	F	Function to write a data area to known master IOlink
axio_pm_r 160()	-	T	F	Function to read a data area from module AXL_F_PM_EF_1F
axio_robj 162()	-	T	F	Function to read objects on fieldbus axioline
axio_wobj 177()	-	T	F	Function to write objects to axioline fieldbus
b0 191() / b0 654()	R	T	F/I	Reads/writes byte 0 of the argument
b1 192() / b1 654()	R	T	F/I	Reads/writes byte 1 of the argument
b2 193() / b2 655()	R	T	F/I	Reads/writes byte 2 of the argument
b3 194() / b3 655()	R	T	F/I	Read/write byte 3 of the argument
b4 194() / b4 655()	R	T	F/I	Read/write byte 4 of the argument
b5 195() / b5 655()	R	T	F/I	Read/write byte 5 of the argument
b6 195() / b6 656()	R	T	F/I	Read/Write byte 6 of the topic
b7 196() / b7 656()	R	T	F/I	Reads/writes byte 7 of the argument
bcc_get_double 197()	R	T	F	Reads a double from a BCC_MSGUSER 793 structure
bcc_get_float 197()	R	T	F	Reads a float from a BCC_MSGUSER 793 structure

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (has a return value)				
Keyword	R	T	F/I	Description
bcc_get_i16 198()	R	T	F	Reads an i16 from a BCC_MSGUSER 793 structure
bcc_get_i32 198()	R	T	F	Reads an i32 from a BCC_MSGUSER 793 structure
bcc_get_i8 199()	R	T	F	Reads an i8 from a BCC_MSGUSER 793 structure
bcc_get_string 199 ()	R	T	F	Reads a string from a BCC_MSGUSER 793 structure
bcc_get_tcp_clients_info 200()	R	T	F	Reads an i32 from a STRU_BCC_TCP_CLIENT 803 structure.
bcc_get_u16 201()	R	T	F	Reads a u16 from a BCC_MSGUSER 793 structure
bcc_get_u32 201()	R	T	F	Reads a u32 from a BCC_MSGUSER 793 structure
bcc_get_u8 202()	R	T	F	Reads a u8 from a BCC_MSGUSER 793 structure
bcc_h_setup 202()	R	T	F	Prepares the structure for the response
bcc_receive 204()	R	T	F	Receives a BCC3 packet
bcc_reply_ack 204()	R	T	F	Sends positive response to received message
bcc_reply_msg 205 ()	R	T	F	Sends response to message received
bcc_reply_nack 206 ()	R	T	F	Sends negative response to message received

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (has a return value)				
Keyword	R	T	F/I	Description
bcc_send 206()	R	T	F	Sends a BCC3 packet
bcc_set_double 656	R	T	I	Writes a double into a BCC_MSGUSER 793 structure
bcc_set_float 657	R	T	I	Writes a float into a BCC_MSGUSER 793 structure
bcc_set_i16 658	R	T	I	Writes an i16 into a BCC_MSGUSER 793 structure
bcc_set_i32 658	R	T	I	Writes an i32 in a BCC_MSGUSER 793 structure.
bcc_set_i8 659	R	T	I	Writes an i8 into a BCC_MSGUSER 793 structure.
bcc_set_string 659	R	T	I	Writes a string to a BCC_MSGUSER 793 structure
bcc_set_u16 660	R	T	I	Writes a u16 into a BCC_MSGUSER 793 structure
bcc_set_u32 660	R	T	I	Writes a u32 in a BCC_MSGUSER 793 structure
bcc_set_u8 661	R	T	I	Writes a u8 into a BCC_MSGUSER 793 structure.
bit_mask_add 661	-	-	I	Adds an hexadecimal value to a variable of type INT
bit_mask_remove 663	-	-	I	Subtracts an hexadecimal value from a variable of type INT
bit_mask_set 665	-	-	I	Initializes a variable of type

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (has a return value)				
Keyword	R	T	F/I	Description
				INT to an hexadecimal value
break 667	R	T	I	Exits a selection block
call 667	R	T	I	Call of a subprogram
case 669	R	T	I	SELECT construct
check_mv_follow2 207()	R	T	F	Controls whether the function mv_follow2 432() can be executed
check_mva_follow2 210()	R	T	F	Controls whether the function mva_follow2 432() can be executed
check_mva_follow2_m 213()	R	T	F	Controls whether the function mva_follow2_m 432() can be executed
cnet_diag()	R	T	F	CNET synchronous communication functions. Reserved
cnet_get_time_sync()	R	T	F	CNET synchronous communication functions. Reserved
cnet_receive()	R	T	F	CNET synchronous communication functions. Reserved
cnet_receive_status()	R	T	F	CNET synchronous communication functions. Reserved
cnet_rq_sync()	R	T	F	CNET synchronous communication functions. Reserved
cnet_send()	R	T	F	CNET synchronous communication functions. Reserved

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (has a return value)				
Keyword	R	T	F/I	Description
cnet_status()	R	T	F	CNET synchronous communication functions. Reserved
co_statusword 246 ()	R	T	F	Returns the contents of the status word of the indicated workstation
co_asw_sdo 215()	R	T	F	Returns the response to the previous function co_send_sdo 242()
co_controlword_state 220()	R	T	F	Function to make transitions in the DS 402 state machine
co_controlword_om 218()	R	T	F	Function to write some bits of the controlword
co_get_ws_diagno 221()	R	T	F	Function to read status information from the indicated workstation
co_par_download 224()	-	T	F	Function to load parameters to the drive by reading them from control CF
co_par_upload 227 ()	-	T	F	Function to save to a file in CF the parameters by reading them from drive
co_rerr 231()	R	T	F	Returns error information of the indicated workstation
co_rnmt 232()	R	T	F	Function to make the NMT operating mode of the indicated workstation
co_robj 234()	R	T	F	Function to read objects on fieldbus

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (has a return value)				
Keyword	R	T	F/I	Description
co_rpdo 241()	R	T	F	Reads a PDO
co_send_sdo 242()	R	T	F	Sends an SDO (Service Data Object) message
co_sync 247()	R	T	F	Enables/disables Sync messages on CANbus
co_wnmt 248()	R	T	F	Imposes the NMT status of the indicated workstation
co_wobj 249()	R	T	F	Function to write objects to fieldbus
co_wpdo 254()	R	T	F	Sends a PDO
coe_robj 255()	R	T	F	Function to read objects on fieldbus COE
coe_wobj 259()	R	T	F	Function to write objects to fieldbus COE
command 264()	R	T	F	Executes a command (directive)
cond_val 265	R	T	I	Simplified IF-THEN-ELSE construct
continue 670	R	T	I	FOR-WHILE construct
cos 265()	R	T	F	Cosine
crc16()	R	T	F	Deprecated. Use crc16_be 266()
crc16_ero()	R	T	F	Deprecated. Use crc16_le 268()
crc16_init()	R	T	F	Deprecated. Use crc16_be 266() / crc16_le 268()
crc16_be 266()	R	T	F	Functions for CRC calculation
crc16_le 268()	R	T	F	Functions for calculating CRC

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (has a return value)				
Keyword	R	T	F/I	Description
ct_cd 670()	R	T	I	Counter down (decremental)
ct_cdr 671()	R	T	I	Counter down ring
ct_cu 672()	R	T	I	Counter up (incremental)
ct_cur 673()	R	T	I	Counter up ring
ct_cv 269() / ct_cv 674()	R	T	F/I	Counter current value
ct_load 674()	R	T	I	Load the CP (preset value) into the CV (current value) of the counter
ct_pv 270() / ct_pv 675()	R	T	F/I	Counter preset value
ct_qd 270()	R	T	F	Counter down outcome
ct_qu 271()	R	T	F	Counter up outcome
ct_reset 675()	R	T	I	Counter CV reset
dec 271()	R	T	F	Decreases the variable
default 675	R	T	I	SELECT construct
delta_enc16 272()	R	T	F	Calculates the difference of two 16-bit quantities by handling the unwrapping
delta_enc32 273()	R	T	F	Calculates the difference of two 32-bit quantities by handling the unwrapping
diff 274()	R	T	F	Returns true if the value of the variable has changed
do 676	R	T	I	DO cycle
dw0 275() / dw0 676()	R	T	F/I	Reads/writes Dword 0 of the

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (has a return value)				
Keyword	R	T	F/I	Description
				argument
dw1 277() / dw1 677()	R	T	F/I	Reads/writes Dword 1 of the argument
dwbe2le 277()	R	T	F	Converts big endian dword (x32) to little endian dword (x32)
dwbe2lef 278()	R	T	F	Converts big endian dword (x32) to little endian float (x32)
dwbe2mbe 278()	R	T	F	Converts big endian dword (x32) to middle big endian dword (x32)
dwbe2mle 279()	R	T	F	Converts big endian dword (x32) to middle little endian dword (x32)
dwell 677	-	T	I	Stops TASK for n seconds
dwle2be 279()	R	T	F	Converts little endian dword (x32) to big endian wdord (x32)
dwle2bef 280()	R	T	F	Converts little endian float (4 bytes) to big endian dword (U32)
dwle2mbe 280()	R	T	F	Converts little endian dword (x32) to middle big endian dword (x32)
dwle2mbef 281()	R	T	F	Converts little endian float (4 bytes) to middle big endian dword (U32)

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (has a return value)				
Keyword	R	T	F/I	Description
dwle2mle 281()	R	T	F	Converts little endian dword (x32) to middle little endian dword (x32)
dwle2mlef 282()	R	T	F	Converts little endian float (4 bytes) to middle little endian dword (U32)
dwmbe2be 282()	R	T	F	Converts middle big endian dword (x32) to big endian dword (x32)
dwmbe2le 283()	R	T	F	Converts middle big endian dword (x32) to little endian dword (x32)
dwmbe2lef 283()	R	T	F	Converts middle big endian (U32) to little endian float (4 bytes)
dwmle2be 284()	R	T	F	Converts middle little endian dword (x32) to big endian dword (x32)
ecat_dev_on_off 284()	-	T	F	Enables/disables a device on EtherCAT line
ecat_get_state 287()	R	T	F	Reads the status of the EtherCAT line
ecat_probe 288()	-	T	F	Reads the information of a device on EtherCAT line
ecat_set_state 290()	R	T	F	Imposes the state of the EtherCAT line
else 677	R	T	I	IF block alternative choice
end_aux 678	R	-	I	End block AUX
end_for 679	R	T	I	End FOR cycle

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (has a return value)				
Keyword	R	T	F/I	Description
end_fun 680	R	T	I	End function
end_motion 681	R	-	I	End block MOTION
end_ref 681	R	-	I	End block REF
end_rule 682	R	-	I	End rule
end_select 682	R	T	I	End SELECT construct
end_while 683	R	T	I	End WHILE construct
end_do_while 679	R	T	I	End DO construct
endif 683	R	T	I	End IF construct
exl_lam 292()	R	T	F	Show included/excluded logical axes of a powerset
excl_lam 684	R	T	I	Include/exclude the logical axes of a powerset
excl_pam 293()	R	T	F	Show the included/excluded physical axes of a powerset
excl_pam 685	R	T	I	Includes/excludes the physical axes of a powerset
f_avail 293()	-	T	F	Memory available on flash
f_close 294()	-	T	F	Close a file opened in flash
f_copy 295()	-	T	F	Copy a file present on the flash
f_del 296()	-	T	F	Delete a file on the flash
f_dir 297()	-	T	F	Information about files in the flash
f_gets 300()	-	T	F	Read from a file in the flash opened in text mode

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (has a return value)				
Keyword	R	T	F/I	Description
f_mkdir 301()	-	T	F	Create a new directory in flash
f_open 302()	-	T	F	Open a file on flash
f_puts 304()	-	T	F	Write a file in the flash opened in text mode
f_read 305()	-	T	F	Reads a file in the flash opened in binary mode
f_read_table 306()	-	T	F	Reads from flash an ascii file and loads the values in it into variables
f_read_table_s 308()	-	t	F	Reads from flash an ascii file and loads the values in it into variables (checks for overflow from the buffer)
f_rename 310()	-	T	F	Renames a file on the flash
f_seek 311()	-	T	F	Moves the pointer inside a file on the flash
f_write 312()	-	T	F	Writes to a file on the flash opened in binary mode
f_format 298()	-	T	F	Formatting flash partitions
fall 314()	R	T	F	Captures the falling edge of a condition
fall_filt 314()	R	T	F	Captures the falling edge of a condition filtered
fill_mem 685	R	T	I	Allows an area of memory to be initialized
filter_lp 317()	R	-	F	Low-pass filter on variable

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (has a return value)				
Keyword	R	T	F/I	Description
filter_lp_st 317()	R	-	F	Low-pass filter on the variable that can be used in a for loop
filter_notch 319()	R	-	F	Notch filter on the variable
filter_notch_st 320()	R	-	F	Low-pass filter on the variable usable in a for loop
first_time 321()	R	-	F	Run only the first time the rule is activated
for 686	R	T	I	FOR loop
force 322()	R	T	F	Force IO
fract 324()	R	T	F	Return fractional part of the argument
function 688()	R	T	I	Function (subroutine)
get_u8 333()	R	T	F	Read a byte (u8) from a buffer at a specified offset
get_i8 329()	R	T	F	Read a byte (i8) from a buffer at a specified offset
get_u16 331()	R	T	F	Read a word (u16) from a buffer at a specified offset
get_i16 328()	R	T	F	Read a word (i16) from a buffer at a specified offset
get_u32 332()	R	T	F	Read a dword (u32) from a buffer at a specified offset
get_i32 328()	R	T	F	Read a dword (i32) from a buffer at a specified offset
get_u64 693()	R	T	I	Read a qword (u64) from a buffer at a specified offset

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (has a return value)				
Keyword	R	T	F/I	Description
get_i64 692()	R	T	I	Read a qword (i64) from a buffer at a specified offset
get_float 325()	R	T	F	Read a float (x32) from a buffer at a specified offset
get_hw_info 326	-	T	F	Get information about cards directly connected to the CPU
get_real 330()	R	T	F	Read a real (x64) from a buffer at a specified offset
get_virt_ow 334()	R	T	F	Read the real state of virtual outputs
goto 693	R	T	I	Label jump instruction
group 694	R	T	I	Activate the rule executors
gv_create 334()	-	T	F	Create a new global variable. Create the information in memory and, if necessary, in retentive memory
gv_datatype 336()	R	T	F	Get the type of a global variable
gv_datatype_byname 337()	R	T	F	Get the type of a global variable
gv_destroy 338()	-	T	F	Remove a global variable. Delete the configuration file and memory space
gv_exist 338()	R	T	F	Check whether the specified string matches a defined global variable
gv_flags 339()	R	T	F	Get the flag associated with a global variable

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (has a return value)				
Keyword	R	T	F/I	Description
gv_flags_byname 341()	R	T	F	Get the flag associated with a global variable
gv_link 342()	R	T	F	Link to an existing global variable. Create an interface to a GV
gv_lock 344()	R	T	F	Block access of the global variable - access is not allowed by any other interface
gv_name 344()	R	T	F	Get the name of a global variable
gv_read 345()	R	T	F	Update the datum of the associated local variable via the interface, reading from the GV
gv_read_partial 345()	R	T	F	Partially update the datum of the associated local variable via the interface, reading from the GV
gv_size 346()	R	T	F	Get the size of a global variable
gv_size_byname 347()	R	T	F	Get the size of a global variable defined by name
gv_unlink 347()	R	T	F	Remove the interface connection to a global variable
gv_unlock 348()	R	T	F	Unlock global variable access- access is then allowed by all interfaces
gv_update_file 348()	-	T	F	Update the information in the configuration file
gv_write 349()	R	T	F	Update the value of the global

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (has a return value)				
Keyword	R	T	F/I	Description
				variable with the value of the associated local variable via the interface
gv_write_partial 349()	R	T	F	Partially update the value of the global variable with the value of the associated local variable via the interface
if 696	R	T	I	IF construct
in_range 350()	R	T	F	Return the value in the given range
inc 351()	R	T	F	Increment the variable
index_of 352()	R	T	F	Return the index of the symbolic variable
inf_report 697	R	T	I	Enable inserting a piece of information into the report
int_after 698	R	T	I	Enable asynchronous delayed rule on event
int_inp 699	R	T	I	Enable asynchronous rule on interrupting input variation
int_timer 701	R	T	I	Enable asynchronous rule on fixed frequency
ipaddr_to_str 353()	R	T	F	Convert a numeric value to a string containing the IP
is_alarm_present 357()	R	T	F	Check the presence of the alarm and parameter in the stack

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (has a return value)				
Keyword	R	T	F/I	Description
is_alarm_par_present 353()	R	T	F	Verify the presence of the alarm in the stack
is_alarm_par_present_us 356()	R	T	F	Verify the presence of the alarm with USV parameter in the stack
is_forced 358()	R	T	F	Verify forced IO
lit 702	R	T	I	Associate a symbolic with a text string
log 362()	R	T	F	Return the natural logarithm of the argument
log10 363()	R	T	F	Return the base10 logarithm of the argument
loop_time 363()	R	T	F	Calculate the time duration between two successive calls
master_filter 364()	R	T	F	Filter the speed of a master
maximum 364()	R	T	F	Return the larger of two values
mbc_connect 365()	-	T	F	Connects the client to the Modbus/TCP server
mbc_init 366()	-	T	F	Initialize Modbus/TCP client
mbc_mask_write_reg 368()	-	T	F	Write registers with mask
mbc_mask_write_reg_s 370()	-	T	F	Write registers with mask via support structure
mbc_read_coils 372()	-	T	F	Read coils
mbc_read_coils_s 373()	-	T	F	Read coils with support structure

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (has a return value)				
Keyword	R	T	F/I	Description
mbc_read_discr_inp 374()	-	T	F	Read discrete inputs
mbc_read_discr_inp_s 376()	-	T	F	Read discrete inputs with support structure
mbc_read_hold_reg 377()	-	T	F	Read holding registers
mbc_read_hold_reg_s 379()	-	T	F	Reading holding registers with support structure
mbc_read_inp_reg 381()	-	T	F	Reading input registers
mbc_read_inp_reg_s 382()	-	T	F	Read input registers with holding structure
mbc_shutdown 384()	-	T	F	Close the connection with the Modbus/TCP server
mbc_write_coils 384()	-	T	F	Write coils
mbc_write_coils_s 386()	-	T	F	Write coils using support structure
mbc_write_multiple_coils 387()	-	T	F	Write coils using Modbus function code of multiple corresponding writes
mbc_write_multiple_coils_s 388()	-	T	F	Write coils via support structure using Modbus function code of multiple corresponding writes
mbc_write_multiple_regs 390()	-	T	F	Write registers using Modbus function code of multiple corresponding writes
mbc_write_multiple_regs_s 392()	-	T	F	Write registers via backing structure using Modbus

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (has a return value)				
Keyword	R	T	F/I	Description
				function code of multiple corresponding writes
mbc_write_reg 393()	-	T	F	Write registers
mbc_write_reg_s 395()	-	T	F	Write registers via support structure
mbcu_mask_write_reg 396()	-	T	F	Write registers with mask by specifying the unitID
mbcu_read_coils 398()	-	T	F	Read coils by specifying the unitID
mbcu_read_discr_inp 400()	-	T	F	Read discrete inputs specifying the unitID
mbcu_read_hold_reg 401()	-	T	F	Read holding registers specifying the unitID
mbcu_read_inp_reg 403()	-	T	F	Read input registers specifying the unitID
mbcu_write_coils 405()	-	T	F	Write coils by specifying the unitID
mbcu_write_multiple_coils 406()	-	T	F	Write coils specifying the unitID using Modbus function code of the corresponding multiple writes
mbcu_write_multiple_regs 408()	-	T	F	Write registers specifying the unitID using Modbus function code of the corresponding multiple writes
mbcu_write_reg 410()	-	T	F	Write registers specifying the

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (has a return value)				
Keyword	R	T	F/I	Description
				unitID
minimum 412()	R	T	F	Returns the smaller of two values
mod 412()	R	T	F	Function that renders the remainder of the division between numeric variables
mot_append 413()	R	T	F	Function for MOT (multiple output table) management
mot_clear 417()	R	T	F	Function for MOT management (multiple output table)
mot_create 417()	R	T	F	Function for MOT handling (multiple output table)
mot_delete 421()	R	T	F	Function for MOT management (multiple output table)
mot_get 421()	R	T	F	Function for MOT management (multiple output table)
mot_remove 423()	R	T	F	Function for MOT management (multiple output table)
motion 703	R	-	I	Start block motion within a rule
move_byte 704	R	T	I	Copy a memory zone
move_mem 705	R	T	I	Copy a memory zone
mt_ats 423()	R	T	F	Atomic test of a variable
mt_ats2 424()	R	T	F	Atomic test of a variable

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (has a return value)				
Keyword	R	T	F/I	Description
mt_dis 706	-	T	I	Enable the execution of a task (time-sharing)
mt_en 706	-	T	I	Disables the execution of a task (time-sharing)
mt_rr_flag 707	-	T	I	Send an infinite OB into execution (loop)
mt_wait 707	-	T	I	Put a TASK on hold
mv_cam 425()	R	-	F	Execute a cubic cam for traits a,b,c)
mv_crimper 427()	R	-	F	Crimper function for flow-pack
mv_follow 428()	R	-	F	Cut on the fly (follow2 recommended)
mv_follow2 432()	R	-	F	Cut on the fly (master and slave not associated with an axis)
mv_mot_exec 445()	R	T	F	Interpolation function using MOT table (multiple output table)
mv_phase_adj 448()	R	-	F	Phase correction between two axes
mv_phase_adj2 448()	R	-	F	Phase correction between two axes
mv_ramp 451()	R	-	F	Deprecated. Use ramp 530()
mv_reach_target 451()	R	-	F	Hooking between two axes
mv_sine_cam 453()	R	-	F	Run a cam (y as a function of x)
mv_synchro 454()	R	-	F	Hooking between two axes
mv_table 455()	R	-	F	Performs a Cam for points

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (has a return value)				
Keyword	R	T	F/I	Description
mv_to 457()	R	-	F	Movement toward a target
mv_to_cj 461()	R	-	F	Movement toward a target with controlled jerk
mv_to_cjv 461()	R	-	F	Movement toward a target with controlled jerk
mv_to_cjv_info 463()	R	T	F	Support function to mv_to_cjv 461()
mv_to_vel 464()	R	-	F	Movement toward a target with a specified final velocity
mv_tracking 466()	R	-	F	Deprecated. Use tracking 622()
mva_follow2 432()	R	-	F	Cut on the fly (only slave associated with an axis)
mva_follow2_m 432()	R	-	F	Cut on the fly (master and slave associated with an axis)
mva_jog 466()	R	-	F	JOG movement (manual)
mva_jog2 468()	R	-	F	JOG movement (manual)
mva_open_loop 470()	R	-	F	Ring opening
mva_to_n 471()	R	-	F	Movement to a target with multiple axes
mva_to_n_cj 475()	R	-	F	Movement toward a target with multiple axes
mva_to_n_v 479()	R	-	F	Movement toward a target with multiple axes also already in motion
mva_zc 486()	R	-	F	Homing movement

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (has a return value)				
Keyword	R	T	F/I	Description
normalize 488()	R	T	F	Returns normalized value between 0 and 1
not 782	R	T	I	Logic operator not
onoff_filt 489()	R	T	F	Captures the filtered rising and falling edges of a condition
or 782	R	T	I	Logic operator or
order 708	R	T	I	Set the order of execution of rule executors
out_m 710	R	T	I	Change the state of the output channels
par_even 491()	R	T	F	Function that makes true if the number of bits in 1 of the byte passed to it is even
par_odd 492()	R	T	F	Function that makes true if the number of bits in 1 of the byte passed to it is odd
planar_comp_ip 494 ()	R	-	F	Function that applies direct position compensation table
planar_comp_cp 493()	R	-	F	Function that applies the inverse position compensation table
pos_capture_cmd 495()	R	T	F	Function to hardware capture the dimension of an axis
pos_capture_sts 499()	R	T	F	Support function to hardware capture the height of an axis
pow 500()	R	T	F	Power elevation

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (has a return value)				
Keyword	R	T	F/I	Description
ps_exclude_lax 511 ()	R	T	F	Exclude the logical axes of a powerset
ps_exclude_pax 513()	R	T	F	Exclude the physical axes of a powerset
ps_reinclude_lax 515()	R	T	F	Include the logical axes of a powerset
ps_reinclude_pax 516()	R	T	F	Include the physical axes of a powerset
ps_power_enable 514()	R	T	F	Powerset (power management) - Enable the power of the powerset
ps_channel_enable 501()	R	T	F	Powerset (power management) - Enable the axes of the power set
ps_channel_reset_fault 507()	R	T	F	Power set (power management) - Send an alarm reset command
ps_drive_reset 509 ()	R	T	F	Powerset (power management) - Send a drive reset command
ps_status 517()	R	T	F	Powerset (power management) - Render information about the status of the power set
ps_channel_enabled 504()	R	T	F	Powerset (power management) - Render information about the enabled axes of a power set
ps_channel_fault 505()	R	T	F	Powerset (power management) - Render information about the faulted axes of a power set

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (has a return value)				
Keyword	R	T	F/I	Description
qwbe2le 711	R	T	I	Convert big endian qword (x64) to little endian qword (x64)
qwbe2lef 521()	R	T	F	Convert big endian qword (x64) to little endian real (x64)
qwbedwx2le 711	R	T	I	Convert dword exchange big endian qword (x64) to little endian qword (x64)
qwbedwx2lef 521()	R	T	F	Convert dword exchange big endian qword (x64) to little endian real (x64)
qwle2be 712	R	T	I	Convert little endian qword (x64) to big endian qword (x64)
qwle2bedwx 712	R	T	I	Convert little endian qword (x64) to dword exchange big endian qword (x64)
qwle2bedwxf 713	R	T	I	Convert little endian real (x64) to dword exchange big endian real (x64)
qwle2bef 714	R	T	I	Convert little endian real (x64) to big endian qword (x64)
qwle2mbe 714	R	T	I	Convert little endian qword (x64) to middle big endian qword (x64)
qwle2mbedwx 712	R	T	I	Convert little endian qword (x64) to dword exchange middle

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (has a return value)				
Keyword	R	T	F/I	Description
				big endian qword (x64)
qwle2mbedwxf 713	R	T	I	Convert little endian real (x64) to dword exchange middle big endian qword (x64)
qwle2mbef 716	R	T	I	Convert little endian real (x64) to middle big endian qword (x64)
qwmbe2le 716	R	T	I	Convert middle big endian qword (x64) to little endian qword (x64)
qwmbe2lef 522 ()	R	T	F	Convert middle big endian qword (x64) to little endian real (x64)
qwmbedwx2le 717	R	T	I	Convert dword exchange middle big endian qword (x64) to little endian qword (x64)
qwmbedwx2lef 522 ()	R	T	F	Convert dword exchange middle big endian qword (x64) to little endian real (x64)
r_and 777	R	T	I	And logical bit to bit
r_not 777	R	T	I	Not logical bit to bit
r_or 777	R	T	I	Or logical bit to bit
r_rol 523 ()	R	T	F	Left-hand rotation of the specified number of bits
r_ror 524 ()	R	T	F	Rotation to the right of the specified number of bits

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (has a return value)				
Keyword	R	T	F/I	Description
r_shl 525()	R	T	F	Shift left of the specified number of bits
r_shr 526()	R	T	F	Shift to the right of the specified number of bits
r_xor 777	R	T	I	Or exclusive bit to bit
r2m16 527()	R	T	F	Real to U16 conversion
r2m32 528()	R	T	F	Real to U32 conversion
r2m8 529()	R	T	F	Real to U8 conversion
ramp 530()	R	T	F	Ramp the magnitude to a target value
rand32 531()	R	T	F	Render a random number (i16)
random 531()	R	T	F	Render a random number (i32)
random_value 531()	R	T	F	Render a random number (real)
range 532()	R	T	F	Render TRUE if the variable is in range
range_adj 533()	R	T	F	Scaling of a variable
rbit 534()	R	T	F	Reset of bits in an integer variable
ref 717	R	-	I	Start Block Reference within a rule
ref_prologue 718()	R	-	I	Special Rule
release 534()	R	T	F	Release forced IO
restart_point 719	-	T	I	Program breakpoint in case of live modification 903
return 719	R	T	I	Stop a function

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (has a return value)				
Keyword	R	T	F/I	Description
rise 536()	R	T	F	Catch the rising edge of a condition
rise_filt 537()	R	T	F	Capture the rising edge of a condition filtered
round 539()	R	T	F	Make the nearest integer
rtc_date 540()	R	T	F	Set the date of the clock calendar
rtc_time 540()	R	T	F	Set the time of the clock calendar
rule 720	R	-	I	Rule definition
rule_epilogue 728	R	-	I	Special rule
rule_freq 729	R	T	I	Sample interval definition
rule_init 730	R	-	I	Initialization rule (runs only at power on)
rule_periodic 730	R	T	I	Run a rule at set frequency (low priority)
rule_prologue 732	R	-	I	Special Rule
safe_appl_ack 541 ()	R	T	F	Set the value of the Application Acknowledge of a safety island
safe_appl_diag 542 ()	R	T	F	Returns the status of the Application Diagnostics of a safety island.
safe_diag 544()	R	T	F	Return the diagnostic log of the desired satellite
safe_diag_reset 546()	R	T	F	Send a Diagnostic Reset to the selected satellite
safe_enab_out 547 ()	R	T	F	Set the value of the selected island's output enable register

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (has a return value)				
Keyword	R	T	F/I	Description
safe_oper_ack 548 ()	R	T	F	Impose an Operation Acknowledge on the selected satellite
safe_prj_info 549()	-	T	F	Enable information regarding a safety project
safe_sts 550()	R	T	F	Return the input/output safety status of the selected satellite
sb0 551() / sb0 733()	R	T	F/I	Return byte 0 of the argument with sign extension
sb1 552() / sb1 733()	R	T	F/I	Return byte 1 of the argument with sign extension
sb2 553() / sb2 733()	R	T	F/I	Return byte 2 of the argument with sign extension
sb3 553() / sb3 734()	R	T	F/I	Return byte 3 of the argument with sign extension
sb4 554() / sb4 734()	R	T	F/I	Return byte 4 of the argument with sign extension
sb5 554() / sb5 734()	R	T	F/I	Return byte 5 of the argument with sign extension
sb6 555() / sb6 735()	R	T	F/I	Return byte 6 of the argument with sign extension
sb7 556() / sb7 735()	R	T	F/I	Return byte 7 of the argument with sign extension
sbit 556()	R	T	F	Set of bits in an integer variable
sdw0 557() / sdw0 735()	R	T	F/I	Return/write the dword 0 of the argument with sign extension

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (has a return value)				
Keyword	R	T	F/I	Description
sdw1 558() / sdw1 736()	R	T	F/I	Return/write the argument's dword 1 with sign extension
search()	R	T	F	Reserved
select 736	R	T	I	Select end_select construct
ser_485_tx_sts 559 ()	R	T	F	Function that renders the status of the last transmission made on the specified device
ser_bcc_h_init 561 ()	R	T	F	Initialize the serial for BCC3 communication between controls
ser_bin_in 562()	R	T	F	Function to read a specified number of characters from the specified device
ser_bin_out 563()	R	T	F	Send characters over the serial channel
ser_bin_ready 563 ()	R	T	F	Report whether there are characters on the serial channel
ser_buff_free 564()	R	T	F	Number of free Bytes on TX buffer
ser_buff_queue 564 ()	R	T	F	Number of occupied Bytes on TX buffer
ser_con 737	R	T	I	Set the parameters of the serial channel
ser_flush 564()	R	T	F	Clean all data in the TX and RX buffers of the specified serial
ser_in 565()	R	T	F	Receive the terminated string from serial

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (has a return value)				
Keyword	R	T	F/I	Description
ser_mod 566()	R	T	F	Set the working mode of the specified serial channel
ser_out 567()	R	T	F	Send the terminated string to the serial
ser_r_sts 567()	R	T	F	Read the status word from the serial
ser_ready 568()	R	T	F	Report the presence of a terminated string on the serial
ser_w_sts 738()	R	T	I	Set the status word of the serial
set_i8 743()	R	T	I	Write a byte (i8) from a buffer at a specified offset
set_u8 747()	R	T	I	Write a byte (u8) from a buffer at a specified offset
set_i16 740()	R	T	I	Write a word (i16) from a buffer at a specified offset
set_u16 745()	R	T	I	Write a word (u16) from a buffer at a specified offset
set_i32 741()	R	T	I	Write a dword (i32) from a buffer at a specified offset
set_u32 746()	R	T	I	Write a dword (u32) from a buffer at a specified offset
set_i64 742()	R	T	I	Write a qword (i64) from a buffer at a specified offset
set_u64 747()	R	T	I	Write a qword (u64) from a buffer at a specified offset

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (has a return value)				
Keyword	R	T	F/I	Description
set_float 739()	R	T	I	Write a float (x32) from a buffer to a specified offset
set_kinematics 569()	R	-	F	Impose kinematic quantities simultaneously
set_real 744()	R	T	I	Write a real (x64) from a buffer at a specified offset
set_version 748()	R	T	I	Let you write the software version of the application
set_virtual_iw 569()	R	T	F	Set the real state of the virtual inputs
shift_left 570()	R	T	F	Shift one bit to the left of the specified memory area
shift_right 571()	R	T	F	Shift one bit to the right of the indicated memory area
shl 573()	R	T	F	Logical shift to the left
shr 573()	R	T	F	Logical shift to the right
sign 574()	R	T	F	Return the sign of the variable
similar 575()	R	T	F	Compare two variables and determines if they are similar
sin 576()	R	T	F	Sine
sizeof 576()	R	T	F	Return the size of the variable
soe_ridn 577()	R	T	F	Function to read parameters on SOE fieldbus
soe_widn 589()	R	T	F	Function to write parameters to fieldbus SOE

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (has a return value)				
Keyword	R	T	F/I	Description
sqrt 594()	R	T	F	Perform square root
statistics 595()	R	T	F	Perform statistics on the indicated variable
str_asc 598()	R	T	F	Convert a character to a number
str_chr 598()	R	T	F	Convert a number to a character
str_compact 599()	R	T	F	Compact spaces
str_equal 600()	R	T	F	Compare two strings
str_find 600()	R	T	F	Search within a string
str_format 601()	R	T	F	Format the string (sprintf)
str_i 602()	R	T	F	Convert an integer numeric value to AscII
str_insert 603()	R	T	F	Insert within a string
str_len 604()	R	T	F	Calculate the length of the string
str_mid 605()	R	T	F	Take in the string
str_r 606()	R	T	F	Convert a real numeric value to AscII
str_replace 607()	R	T	F	Substitute within a string
str_val 608()	R	T	F	Render, in numeric format, a value contained in a string
str_to_ipaddr 608()	R	T	F	Convert a string containing the IP to a numeric value
store_nvr_nopf 595()	R	T	F	Save the specified NVRs in static memory

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (has a return value)				
Keyword	R	T	F/I	Description
store_nvrr_nopf 596()	R	T	F	Save the NVRs specified in static memory
sum8_mem 611()	R	T	F	Return the sum of n elements of a U8 array
sum16_mem 611()	R	T	F	Return the sum of n elements of an array of U16
sum32_mem 612()	R	T	F	Return the sum of n elements of an array of U32
sumr_mem 612()	R	T	F	Return the sum of n elements of an array of real
sw0 612() / sw0 749()	R	T	F/I	Return/write the word 0 of the argument with the sign extension
sw1 613() / sw1 749()	R	T	F/I	Return/write the word 1 of the argument with the sign extension
sw2 614() / sw2 750()	R	T	F/I	Return/write the word 2 of the argument with the sign extension
sw3 614() / sw3 750()	R	T	F/I	Return/write the word 3 of the argument with the sign extension
tan 615()	R	T	F	Tangent
terminator 750	R	T	I	Insert a terminator
time_micro 616()	R	T	F	Return the time in microseconds since the axis control was turned on
tm_et 616() / tm_et 751()	R	T	F/I	Read/set the timer time
tm_int 751()	R	T	I	Integral timer
tm_pt 617() / tm_pt 756()	R	T	F/I	Read/set the timer preset

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (has a return value)				
Keyword	R	T	F/I	Description
tm_pt2 618() / tm_pt2 756()	R	T	F/I	Read/set the second timer preset (timer square)
tm_q 619()	R	T	F	Timer output (coil)
tm_reset 756()	R	T	I	Reset the timer time (integral timer)
tm_sqw 756()	R	T	I	Timer square wave
tm_of 753()	R	T	I	Timer OFF
tm_on 754()	R	T	I	Timer ON
tm_tp 758()	R	T	I	Timer pulse
tobcd 619()	R	T	F	Convert the value from binary to BCD
tobin 620()	R	T	F	Convert value from BCD to binary
todeg 621()	R	T	F	Convert value from radians to degrees
tolower 621()	R	T	F	Convert one character to lower case
torad 621()	R	T	F	Convert value from degrees to radians
toupper 622()	R	T	F	Convert a character to uppercase
tracking 622()	R	T	F	Bring the controlled magnitude to the specified target
trunc 624()	R	T	F	Return the integer part of the argument
udp_open_client 626()	R	T	F	Open an udp client connection
udp_open_server 628()	R	T	F	Open an udp server connection
udp_close 625()	R	T	F	Close an udp connection

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (has a return value)				
Keyword	R	T	F/I	Description
udp_send 636()	R	T	F	Send an udp data packet
udp_send_to 640()	R	T	F	Send an udp data packet
udp_recv 631()	R	T	F	Receive an udp data packet
udp_recv_from 632()	R	T	F	Receive an udp data packet
udp_send_notify 638()	R	T	F	Send an udp data packet with notification
udp_recv_notify 634()	R	T	F	Receive an udp data packet with notification
virt_io 642()	-	T	F	Define a virtual I/O area
w0 644() / w0 759()	R	T	F/I	Read/write word 0 of the topic
w1 645() / w1 759()	R	T	F/I	Read/write word 1 of the topic
w2 645() / w2 760()	R	T	F/I	Read/write word 2 of the topic
w3 646() / w3 760()	R	T	F/I	Read/write word 3 of the topic
wbe2le 647()	R	T	F	Convert big endian word (x16 or x32) to little endian word (x16)
wle2be 648()	R	T	F	Convert little endian word (x16 or x32) to big endian word (x16)
wbit 647()	R	T	F	Set/Reset bits to an integer variable
while 760	R	T	I	While end_while construct
write_dis 648()	R	T	F	Write a string to the 4*20 display
xor 782	R	T	I	Logical or exclusive operator

© 2025 Robox SpA

Functions error codes

List of available codes.

Code	Label	Description
1	EPERM	Operation not allowed
2	ENOENT	No files or directories of this type
3	ESRCH	No process of this type
4	EINTR	System call aborted
5	EIO	I/O error
6	ENXIO	No devices or addresses of this type
7	E2BIG	List of arguments too long
8	ENOEXEC	Execution format error
9	EBADF	Wrong file number
10	ECHILD	No child process
11	EAGAIN	Try again
12	ENOMEM	Out of memory
13	EACCES	Permissions denied
14	EFAULT	Wrong address
15	ENOTBLK	Block device required
16	EBUSY	Device or resource busy
17	EEXIST	File exists
18	EXDEV	Link between devices
19	ENODEV	Device does not exist
20	ENOTDIR	It is not a directory
21	EISDIR	It is a directory
22	EINVAL	Invalid argument
23	ENFILE	File table overflow
24	EMFILE	Too many opened files
25	ENOTTY	It is not a typewriter
26	ETXTBSY	Text file busy
27	EFBIG	File too large

© 2025 Robox SpA

Code	Label	Description
28	ENOSPC	No more space on the device
29	ESPIPE	Illegal search
30	EROFS	Read-only file system
31	EMLINK	Too many links
32	EPIPE	Broken pipe
33	EDOM	Mathematical argument outside the domain of the function
34	ERANGE	Mathematical result not representable
35	EDEADLK	Resource deadlock would occur
36	ENAMETOOLONG	File name too long
37	ENOLCK	No record lock available
38	ENOSYS	Function not implemented
39	ENOTEMPTY	Directory not empty
40	ELOOP	Too many symbolic links encountered
42	ENOMSG	No message of desired type
43	EIDRM	Identifier removed
44	ECHRNG	Channel number out of range
48	ELNRNG	Link number out of range
49	EUNATCH	Protocol driver not connected
53	EBADR	Invalid request descriptor
56	EBADRQC	Invalid request code
57	EBADSLT	Invalid slot
60	ENOSTR	Device is not a stream
61	ENODATA	No data available
62	ETIME	Timer expired
63	ENOSR	Stream resources exhausted
64	ENONET	Machine is not on the network
65	ENOPKG	Package not installed

© 2025 Robox SpA

Code	Label	Description
66	EREMOTE	Object is remote
67	ENOLINK	Connection has been broken
68	EADV	Advertisement error
69	ESRMNT	Assembly error
70	ECOMM	Communication error when sending
71	EPROTO	Protocol error
72	EMULTIHOP	Multihop attempt
73	EDOTDOT	RFS-specific error
74	EBADMSG	Not a data message
75	EOVERFLOW	Value too large for defined data type
76	ENOTUNIQ	Name not unique on the network
77	EBADFD	File descriptor in bad state
78	EREMCHG	Remote address changed
79	ELIBACC	Unable to access a needed shared library
80	ELIBBAD	Accessing a damaged shared library
81	ELIBSCN	Corrupted .lib section in a.out
82	ELIBMAX	Attempt to link too many shared libraries
83	ELIBEXEC	Unable to directly execute a shared library
84	EILSEQ	Illegal byte sequence
85	ERESTART	Broken system call needs to be restarted
86	ESTRPIPE	Stream pipe error
87	EUSERS	Too many users
88	ENOTSOCK	Socket operation on non-pocket
89	EDESTADDRREQ	Destination address required
90	EMGSIZE	Message too long

© 2025 Robox SpA

Code	Label	Description
91	EPROTTYPE	Wrong type protocol for socket
92	ENOPROTOOPT	Protocol not available
93	EPROTONOSUPPORT	Protocol not supported
94	ESOCKTNOSUPPORT	Socket type not supported
95	EOPNOTSUPP	Unsupported operation on transport endpoint
96	EPFNOSUPPORT	Protocol family not supported
97	EAFNOSUPPORT	Protocol address family not supported
98	EADDRINUSE	Address already in use
99	EADDRNOTAVAIL	Unable to assign requested address
100	ENETDOWN	Network is down
101	ENETUNREACH	Network is unreachable
102	ENETRESET	Network has broken the connection due to a reset
103	ECONNABORTED	Connection interruption caused by software
104	ECONNRESET	Connection reset by the peer
105	ENOBUFS	No buffer space available
106	EISCONN	Transport endpoint is already connected
107	ENOTCONN	Transport endpoint is not connected
108	ESHUTDOWN	Unable to send after transport endpoint stopped
109	ETOOMANYREFS	Too many references: unable to splice
110	ETIMEDOUT	Connection expired
111	ECONNREFUSED	Connection refused
112	EHOSTDOWN	Host is down
113	EHOSTUNREACH	No path to host
114	EALREADY	Operation already in progress
115	EINPROGRESS	Operation in progress

© 2025 Robox SpA

Code	Label	Description
116	ESTALE	NFS file handle obsolete
117	EUCLEAN	Structure needs to be cleaned up
120	EISNAM	It is a file of type named
121	EREMOTEIO	Remote I/O error
122	EMSGSIZE	Message too long
123	ENOMEDIUM	No support found
124	EMEDIUMTYPE	Wrong type of support

Return codes of the MV_ and MVA_ functions

List of return codes common to axis motion management functions and instructions.

Symbolic	Value	Meaning
M_IN_ALARM	-6	Warning situation present
M_ILLEGAL_ARGS	-5	Illegal argument(s)
M_LOW_ACCELERATION	-4	Insufficient acceleration
M_SLAVE_EARLY_STOPPED	-3	Slave in advance, stopped
M_SLAVE_IS_EARLY	-2	Slave in advance to appointment
M_SLAVE_TOO_LATE	-1	Slave delay to appointment
M_INITIALIZING	1	Initialization function
M_POSITIONING	2	Initial positioning
M_WAITING	3	Waiting for start
M_WAITING_NEXT_CYCLE	4	Waiting for next cycle
M_FOLLOWING	5	Following in progress
M_REACHED	6	Target reached/mission terminated
M_HOMING	7	Cycle start position recovery/resetting in progress
M_SYNCHRONIZED	8	Speed synchronized to master
M_ACCEL_LIMITED	9	Acceleration limited to desired max
M_SPEED_LIMITED	10	Speed limited to desired max

© 2025 Robox SpA

Symbolic	Value	Meaning
M_ADJUSTING_UP	11	Correction in progress (triangle ascent)
M_ADJUSTING_DOWN	12	Correction in progress (triangle descent)
M_OVERSHOOTING	13	Target position is/will be exceeded
M_HOLDED	14	Movement stopped due to HOLD flag. Also returned in cases where the desired axis speed is set to zero (e.g. the *dSpeed* field of struct stru_mvtocjv 857) or the feedrate is set to zero
M_ACCEL	15	Acceleration in progress
M_AT_SPEED	16	Movement speed reached
M_DECEL	17	Deceleration in progress
M_LAST_CYCLE	18	Last cycle before passing *pDes*
M_OUT_OF_LIMITS	19	Out of set limits

Modbus error codes

Modbus error codes

Modbus error code	Value	Description
MBC_WAIT	1	Command waiting for response
MBC_OK	0	Command executed
MBC_INVAL	-1	Invalid arguments
MBC_BUSY	-2	Resource occupied
MBC_PERM	-3	Permission denied
MBC_CONN	-4	Connection failed
MBC_NOBUFS	-5	Buffer exhausted
MBC_NOMEM	-6	Insufficient memory
MBC_TX_ERR	-7	Transmission error
MBC_RX_ERR	-8	Receive error
MBC_TIMEOUT	-9	Timeout blocking command

© 2025 Robox SpA

Modbus error code	Value	Description
MBC_EXC_CODE_1	-10	Response with exception 1 (function code not supported by the server)
MBC_EXC_CODE_2	-11	Response with exception 2 (invalid address)
MBC_EXC_CODE_3	-12	Response with exception 3 (invalid quantity or value)
MBC_EXC_CODE_4	-13	Response with exception 4 (operation failed)
MBC_WRONG_ANSWER	-14	Invalid response
MBC_INVAL_MBC	-15	Invalid MBC id

SDO Error Codes (CoE/SoE)

SDO CoE/SoE Error Codes

SDO error code	Meaning
0x0500 0000	General SDO protocol error found.
0x0503 0000	Toggle bit not toggled.
0x0504 0000	SDO protocol timeout.
0x0504 0001	Invalid or unknown client/server command identifier.
0x0504 0002	Invalid block size (blocking management only).
0x0504 0003	Invalid sequence number (blocking management only).
0x0504 0004	CRC error (blocking management only).
0x0504 0005	Memory overflow.
0x0601 0000	Unsupported access to an object.
0x0601 0001	Attempt to read a write-only object.
0x0601 0002	Attempt to write a read-only object.
0x0602 0000	Object does not exist in the object dictionary.
0x0604 0041	Unable to map the object to the PDO.
0x0604 0042	The number and length of objects to be mapped would exceed the length of the PDO.

© 2025 Robox SpA

SDO error code	Meaning
0x0604 0043	Incompatibility of parameters.
0x0604 0047	Incompatibility internal to the device.
0x0606 0000	Failed access due to hardware error.
0x0607 0010	Data type does not match, length of service parameter does not match.
0x0607 0012	Data type does not match, length of service parameter is too high.
0x0607 0013	Data type does not match, service duration parameter is too low.
0x0609 0011	Subindex does not exist.
0x0609 0030	Parameter value range exceeded (write access only).
0x0609 0031	Parameter value written too high.
0x0609 0032	Value of the written parameter too low.
0x0609 0036	Maximum value is less than minimum value.
0x060A 0023	Resource not available: SDO connection.
0x0800 0000	General error.
0x0800 0020	Data cannot be transferred or stored in the application.
0x0800 0021	Data cannot be transferred or stored in the application due to local control.
0x0800 0022	Data cannot be transferred or stored in the application due to the current state of the device.
0x0800 0023	Dynamic object dictionary generation fails or no object dictionary is present (e.g., object dictionary is generated from a file and generation fails due to a file error).
0x0800 0024	No data available.
Other values	Invalid code - no CiA DS301 v4.1 is present.

SDO SoE Error Codes

SDO error code	Meaning
0x0009	Invalid access to element 0.
0x1001	Does not exist.
0x1009	Invalid access to element 1.

© 2025 Robox SpA

SDO error code	Meaning
0x2001	Name does not exist.
0x2002	Name undersized in transmission.
0x2003	Name oversized in transmission.
0x2004	Name cannot be changed.
0x2005	Name currently write-protected.
0x3002	Attribute undersized in transmission.
0x3003	Attribute oversized in transmission.
0x3004	Attribute cannot be changed.
0x3005	Attribute currently write-protected.
0x4001	Unit does not exist.
0x4002	Unit undersized in transmission.
0x4003	Unit oversized in transmission.
0x4004	Unit cannot be changed.
0x4005	Unit currently write-protected.
0x5001	Minimum value in input nonexistent.
0x5002	Minimum value in input undersized in transmission.
0x5003	Minimum value in input oversized in transmission.
0x5004	Minimum value in input cannot be changed.
0x5005	Minimum value in input currently write-protected.
0x6001	Maximum value in input nonexistent.
0x6002	Maximum value in input undersized in transmission.
0x6003	Maximum value in input oversized in transmission.
0x6004	Maximum value in input cannot be changed.
0x6005	Maximum value in input currently write-protected.
0x7001	Element does not exist.
0x7002	Undersized element in transmission.
0x7003	Oversized element in transmission.

© 2025 Robox SpA

SDO error code	Meaning
0x7004	Element cannot be changed.
0x7005	Element currently write-protected.
0x7006	Element below *minimum input value* limit.
0x7007	Element above limit of *maximum input value*.
0x7008	Element not corrected.
0x7009	Element protected by password.
0x700A	Element temporarily not editable (AT or MDT).
0x700B	Indirectly invalid.
0x700C	Temporarily non-modifiable element (parameter or operating mode).
0x7010	Command already active.
0x7011	Command not interruptible.
0x7012	Command not available (at this stage).
0x7013	Command not available (invalid parameter).

Zero-cycles types

Types of zero cycle:

- Linear axis: zero cycle towards the micro 66
- Linear axis: zero cycle towards the micro + index 69
- Linear axis: zero cycle towards the end of stroke 72
- Linear axis: zero cycle towards the end of stroke + index 74
- One-way rotary axis: zero cycle towards the micro + index with preliminary search for the micro variation 76
- One-way rotary axis: zero cycle towards the micro + index without preliminary search for the micro variation 80
- One-way rotary axis: zero cycle towards the micro with preliminary search for the micro variation 83
- One-way rotary axis: zero cycle towards the micro without preliminary search for the micro variation 85

Linear axis: zero cycle towards the micro

Validity: **Rule**

Case 1: the axis starts from the PS point in area A

© 2025 Robox SpA

```
                                |---------------------------|
            A                   |            B              | Mechanic cam read by micro
 |-------------------------|                                |

 |-----------------------------------------------------| Opeartive field of the axes
 ^      ^       ^          ^   ^     ^                 ^
 |      |       |          |   |     |                 |
 |      PS      P2         |   PF    P1                |
 |                         |                           |
Qmin                  Preset quote                    Qmax

        |---------------->------------| Preliminary mission 1 (+velPos)
                |-------<------------| Preliminary mission 2 (-velPos)
                |------->------|       Zero cycle mission (+velZC)
```

Phases:

- The state of the micro is read in order to define the sign of the movement speed. Assuming parameter *zcFlag*.0 = 0, since if the axis is in zone **A** the velocity contained in parameter *velPos* will be assumed positive.
- Preliminary Mission 1: The axis moves at the velocity (+*velPos*) until it encounters the rising edge of the micro, then decelerates and stops at position **P1**.
- Preliminary Mission 2: The axis moves at the speed (-*velPos*) until it encounters the falling edge of the micro, then decelerates and stops at position **P2**.
- Final Mission (CYCLE0): The axis performs the final movement with a maximum stroke equal to the value of the *spSch* parameter, with the new speed set (+*velZc*).
  When the micro rising edge is found, the mission is stopped and the axis stops in **PF**.
- The new actual real position is assigned by the value obtained by summing the delta position (**PF**-quote read on the rising edge of the micro) to the *Preset* parameter.
- The predefined variable c0_done is updated by setting to 1 the axis bit.

Legend:

- *A, area where the micro reads 0*
- *B, area where the micro reads 1*
- *PS, initial start point for mva_zc 486()*
- *PF, target stop point for mva_zc 486()*
- *P1, intermediate point 1*
- *P2, intermediate point 2*
- *Qmin, axis minimum position*
- *Qmax, axis maximum position*

Case 2: the axis starts from the PS point in area B

© 2025 Robox SpA

```
                                |---------------------------|
          A                     |           B               | Mechanic cam read by micro
  |-------------------------|                               |

  |-----------------------------------------------------| Operative field of the axis
  ^              ^           ^   ^     ^                 ^
  |              |           |   |     |                 |
  |             P1           |  PF    PS                 |
  |                          |                           |
Qmin                  Preset quote                      Qmax

                 |-------<------------| Preliminary mission 1 (-velPos)
                 |------->------|       Zero cycle mission (+velZC)
```

Phases:

- The state of the micro is read in order to define the sign of the movement speed. Assuming parameter *zcFlag*.0 = 0, since if the axis is in zone **B**, the velocity contained in parameter *velPos* will be assumed negative.
- Preliminary Mission 1: The axis moves at the velocity (-*velPos*) until it encounters the falling edge of the micro, then decelerates and stops at position **P1**.
- Final Mission (CYCLE0): The axis performs the final movement, whose max stroke is equal to the value of the parameter *spSch*, with the new programmed speed (+*velZc*). When the micro rising edge is found, the mission is stopped and the axis stops in **PF**.
- The new actual real position is assigned by the value obtained by summing the delta position (**PF**-quote read on the rising edge of the micro) to the *Preset* parameter.
- The predefined variable c0_done is updated by setting to 1 the axis bit.

Legend:

- *A, area where the micro reads 0*
- *B, area where the micro reads 1*
- *PS, initial start point for mva_zc 486 ()*
- *PF, target stop point for mva_zc 486 ()*
- *P1, intermediate point 1*
- *P2, intermediate point 2*
- *Qmin, axis minimum position*
- *Qmax, axis maximum position*

Examples

Example of use:

```
INT result
STRU_MVZC 863 zc
INT Flag
...
rule 123
    axes 1
    motion
        ; First time initialization
        if (first_time 321 ())
            Flag = true
            zc.nAx     = 1 ; Axis number
```

```
            zc.preset = 0 ; Preselection quotas
            zc.velPos = 10.0 ; Epos(nAx) threshold in units to
consider end of stroke reached (if you also have cv(nAx)
<(zc.velZC/1000.0) )
            zc.velZC  = rVelc0 ; Speed for final handling CYCLE0
mva_zc
            zc.acc    = rMaxAcc * 0.5 ; Acceleration to be used in
all movements
            zc.jerk   = rMaxAcc * 10 ; Derivative of acceleration
            zc.spSCh  = 4096 ; Maximum space of the final movement
CYCLE0
            zc.filter = 0 ; If drive in canopen reappears % torque
reduction
            zc.zcFlag.0 = 1 ; Positive speed
            zc.zcFlag.1 = 0 ; You select the falling edge of the
micro
            zc.zcFlag.2 = 0 ; Disable full precision
            zc.zcFlag.3 = 0 ; Enable use of the micro
            zc.zcFlag.4 = 0 ; The execution of the movement is
bidirectional
            zc.zcFlag.5 = 0 ; The space that can be traversed during
preliminary handling is unlimited
            zc.zcFlag.6 = 0 ; Not used
            zc.zcFlag.7 = 0 ; Disable search of the end of stroke
            zc.zcFlag.8 = 0 ; Disable 0 cycle ON_POSITION
            zc.zcFlag.9 = 0 ; Disable positioning on preset elevation
            zc.zcFlag.10 = 0 ; Disable repositioning within software
end of stroke switches.
        endif

        ; Call the move function
        result = mva_zc 486(Flag, zc)
    end_motion
end_rule
```

Linear axis: zero cycle towards the micro + index

Validity: **Rule**

Case 1: the axis starts from the PS point in area A

```
                                |---------------------------|
          A                     |            B              | Mechanic cam read by micro
 |-------------------------|                                |

                                CH0
                                 |
 |-----------------------------------------------------| Operative field of the axis
 ^      ^       ^             ^     ^     ^             ^
 |      |       |             |     |     |             |
 |      PS      P2            |     PF    P1            |
 |                            |                         |
Qmin                    Preset quote                    Qmax

        |---------------->----------------| Preliminary mission 1 (+velPos)
                |-------<----------------| Preliminary mission 2 (-velPos)
                |------->----------|       Zero cycle mission (+velZC)
```

Phases:

- The state of the micro is read in order to define the sign of the movement speed. Assuming parameter *zcFlag*.0 = 0, since if the axis is in zone **A** the velocity contained in parameter *velPos* will be assumed positive.
- Preliminary Mission 1: The axis moves at the velocity (+*velPos*) until it encounters the rising edge of the micro, then decelerates and stops at position **P1**.
- Preliminary Mission 2: The axis moves at the speed (-*velPos*) until it encounters the falling edge of the micro, then decelerates and stops at position **P2**.
- Final Mission (CYCLE0): The axis performs the final movement with a maximum stroke equal to the value of the *spSch* parameter, with the new speed set (+*velZc*).
  When the first **CH0** encoder is found with the micro in 1, the mission is stopped and the axis stops in **PF**.
- The new actual real position is assigned by the value obtained by summing the delta position (**PF**-quote read on the first **CH0** encoder with the micro in 1) to the *Preset* parameter.
- The predefined variable c0_done is updated by setting to 1 the axis bit.

Legend:

- *A, area where the micro reads 0*
- *B, area where the micro reads 1*
- *PS, initial start point for mva_zc 486()*
- *PF, target stop point for mva_zc 486()*
- *P1, intermediate point 1*
- *P2, intermediate point 2*
- *Qmin, axis minimum position*
- *Qmax, axis maximum position*
- *CH0, 0 channel of the incremental encoder*

Case 2: the axis starts from the PS point in area B

© 2025 Robox SpA

```
                          |---------------------------|
          A               |           B               | Mechanic cam read by micro
 |-------------------------|                           |

                          CH0
                           |
 |-----------------------------------------------------| Operative field of the axis
 ^              ^           ^   ^     ^               ^
 |              |           |   |     |               |
 |             P1           |   PF    PS              |
 |                          |                         |
Qmin                  Preset quote                   Qmax

                |-------<--------------| Preliminary mission 1 (-velPos)
                |------->--------|       Zero cycle mission (+velZC)
```

Phases:

- The state of the micro is read in order to define the sign of the movement speed. Assuming parameter *zcFlag*.0 = 0, since if the axis is in zone **A** the velocity contained in parameter *velPos* will be assumed negative.
- Preliminary Mission 1: The axis moves at the velocity (-*velPos*) until it encounters the falling edge of the micro, then decelerates and stops at position **P1**.
- Final Mission (CYCLE0): The axis performs the final movement with a maximum stroke equal to the value of the *spSch* parameter, with the new speed set (+*velZc*).
  When the first **CH0** encoder is found with the micro in 1, the mission is stopped and the axis stops in **PF**.
- The new actual real position is assigned by the value obtained by summing the delta position (**PF**-quote read on the first **CH0** encoder with the micro in 1) to the *Preset* parameter.
- The predefined variable c0_done is updated by setting to 1 the axis bit.

Legend:

- *A, area where the micro reads 0*
- *B, area where the micro reads 1*
- *PS, initial start point for mva_zc 486 ()*
- *PF, target stop point for mva_zc 486 ()*
- *P1, intermediate point 1*
- *P2, intermediate point 2*
- *Qmin, axis minimum position*
- *Qmax, axis maximum position*
- *CH0, 0 channel of the incremental encoder*

Examples

Example of use:

```
INT result
STRU_MVZC 863  zc
INT Flag
...
rule 123
    axes 1
    motion
        ; First time initialization
        if (first_time 321 ())
```

```
            Flag = true
            zc.nAx    = 1 ; Axis number
            zc.preset = 0 ; Preselection quotas
            zc.velPos = rVelc0 ; Speed for preliminary movements
mva_zc
            zc.velZC  = rVelc0 * 0.5 ; Speed for final movement
CYCLE0 mva_zc
            zc.acc    = rMaxAcc * 0.5 ; Acceleration to be used in
all movements
            zc.jerk   = rMaxAcc * 10 ; Derivative of acceleration
            zc.spSCh  = 20000 ; Maximum space of the final movement
CYCLE0
            zc.filter = 0 ; Time (sec) of micro signal filtering
            zc.zcFlag.0 = 0  ; Negative speed when the micro is in 1
            zc.zcFlag.1 = 1  ; Wants micro in 1 when looking for CH0
            zc.zcFlag.2 = 1  ; Enable full-precision (in this case
CH0 since transducer enc_inc)
            zc.zcFlag.3 = 0  ; Enable use of micro
            zc.zcFlag.4 = 0  ; Bidirectional search (since the axis
is linear)
            zc.zcFlag.5 = 0  ; Preliminary unrestricted excursion
missions
            zc.zcFlag.6 = 0  ; Not used (because two-way search)
            zc.zcFlag.7 = 0  ; Disable end of troke search (because
micro enabled)
            zc.zcFlag.8 = 0  ; Disable zero cycle ON_POSITION
            zc.zcFlag.9 = 0  ; Disable positioning on preset
elevation
            zc.zcFlag.10 = 0 ; Disable repositioning within software
limit switches.
        endif

        ; Call the move function
        esito = mva_zc 486(Flag, zc)
    end_motion
end_rule
```

Linear axis: zero cycle towards the end of stroke

Validity: **Rule**

Unique case, the axis starts from the PS point

```
    |-------------------------------------------|    Operative field of the axis
    ^                      ^      ^             ^
    |                      |      |             |
    |                      PS     PF            |
    |                                           |
   Qmin                                        Qmax
Min stroke                                   Max stroke
                                                ^
                                                |
                                             Preset quote

                          |-------->----------|  Zero cycle mission (+velZC)
                                  |-----<------|  Moves away from stroke (-velZC)
```

Phases:

- The servo alarm (SAM) of the axis is disabled.
- CYCLE0 mission (search for the end of stroke): the axis moves at the speed (+*velZC*) until the conditions "abs(CV) < zc.*velZC*/1000.0" and "abs(IP-CP) > zc.*velPos*" are met.
  NOTE: Since RTE v34.24.10, if *zcFlag*.13 (0x2000) of stru_mvzc 863 is set at 1, the condition "abs(IP-CP) > zc.*velPos*" is disabled.
- The previous value is set again on the SAM mask related to the axis.
- The axis moves away from the end of stroke: the axis moves at the velocity (-*velZC*) until it travels a space at least equal to zc.*spSCh* and stops at the **PF** position.
- The value obtained by subtracting the value of the **PF** point from the *Preset* parameter is assigned as the new real quote.
- The predefined variable c0_done is updated by setting the axis related bit to 1.

Legend:

- *PS, initial start point for mva_zc 486()*
- *PF, target stop point for mva_zc 486()*
- *Qmin, axis minimum position (min_str)*
- *Qmax, axis maximum position (max_str)*

Examples

Example of use:

```
INT result
STRU_MVZC 863 zc
INT Flag
...
rule 123
    axes 1
    motion
        ; First time initialization
        if (first_time 321())
            Flag = true
            zc.nAx    = 1 ; Axis number
            zc.preset = 0 ; Preselection quotas
```

```
            zc.velPos = 10.0 ; Threshold epos(nAx) in units x
consider paracart reached (if you also have cv(nAx)<(zc.velZC/1000.0)
)
            zc.velZC  = rVelc0 ; Speed for final movement CYCLE0
mva_zc
            zc.acc    = rMaxAcc * 0.5 ; Acceleration to be used in
all movements
            zc.jerk   = rMaxAcc * 10 ; Derivative of acceleration
            zc.spSCh  = 20.0 ; Minimal space x mission away from the
paracart (unit).
            zc.filter = 0 ; If drive in canopen represents % torque
reduction
            zc.zcFlag.0 = 1  ; Positive speed x paracart search
            zc.zcFlag.1 = 0  ; Torque reduction %, only if drive type
managed.
            zc.zcFlag.2 = 0  ; Disabilita full-precision (ch0 enc)
            zc.zcFlag.3 = 1  ; Disable use of micro (because paracar
research is enabled)
            zc.zcFlag.4 = 1  ; Not used
            zc.zcFlag.5 = 1  ; Preliminary missions of limited
excursion (equal to max_str(nAx)-min_str(nAx))
            zc.zcFlag.6 = 0  ; Not used
            zc.zcFlag.7 = 1  ; Enabled end of stroke search
            zc.zcFlag.8 = 0  ; Disable zero cycle ON_POSITION
            zc.zcFlag.9 = 0  ; Disable positioning on preset
elevation
            zc.zcFlag.10 = 0 ; Disable repositioning within software
limit switches.
        endif

        ; Call the move function
        result = mva_zc 486(Flag, zc)
    end_motion
end_rule
```

Linear axis: zero cycle towards the end of stroke + index

Validity: **Rule**

Unique case, the axis starts from the PS point

```
                                        CH0
                                         |
   |-------------------------------------------| Operative field of the axis
   ^                      ^      ^   ^        ^
   |                      |      |   |        |
   |                     PS     PF   |        |
   |                                |        |
  Qmin                         Preset quote  Qmax
Min stroke                                  Max stroke

                         |-------->----------|  Preliminary mission (+velZC)
                                |-----<------|  Zero cycle mission (-velZC)
```

Phases:

- The servo alarm (SAM) of the axis is disabled.
- Preliminary mission (search for the end of stroke): the axis moves at the speed (+*velZC*) until the conditions "abs(CV) < zc.*velZC*/1000.0" and "abs(IP-CP) > zc.*velPos*" are met.
  NOTE: Since RTE v34.24.10, if *zcFlag*.13 (0x2000) of stru_mvzc 863 is set at 1, the condition "abs(IP-CP) > zc.*velPos*" is disabled.
- The previous value is set again on the SAM mask related to the axis.
- CYCLE0 final mission: the axis moves at the velocity (-*velZC*) until it reach the **CH0** encoder, then decelerates and stops at the **PF** position.
- The value obtained by adding the position delta (**PF**-value read on the first **CH0**) to the *Preset* parameter, is assigned as the new real quote.
- The predefined variable c0_done is updated by setting the axis related bit to 1.

Legend:

- *PS, initial start point for mva_zc 486()*
- *PF, target stop point for mva_zc 486()*
- *Qmin, axis minimum position (min_str)*
- *Qmax, axis maximum position (max_str)*
- *CH0, 0 channel of the incremental encoder*

Examples

Example of use:

```
INT result
STRU_MVZC 863 zc
INT Flag
...
rule 123
    axes 1
    motion
        ; First time initialization
        if (first_time 321())
            Flag = true
            zc.nAx     = 1 ; Axis number
```

```
            zc.preset = 0 ; Preselection quotas
            zc.velPos = 10.0 ; Threshold epos(nAx) in units to
consider paracart achieved (if you also have cv(nAx)
<(zc.velZC/1000.0) )
            zc.velZC  = rVelc0 ; Speed for final movement CYCLE0
mva_zc 486
            zc.acc    = rMaxAcc * 0.5 ; Acceleration to be used in
all movements
            zc.jerk   = rMaxAcc * 10 ; Derivative of acceleration
            zc.spSCh  = 4096 ; Maximum space of the final movement
CYCLE0
            zc.filter = 0 ; If drive in fieldbus reappears % torque
reduction
            zc.zcFlag.0 = 1  ; Positive speed x paracart search
            zc.zcFlag.1 = 0  ; Not used
            zc.zcFlag.2 = 1  ; Abilita full-precision (ch0 enc)
            zc.zcFlag.3 = 1  ; Disable use of micro (because end of
stroke search is enabled)
            zc.zcFlag.4 = 1  ; Not used
            zc.zcFlag.5 = 1  ; Preliminary missions of limited
excursion (equal to max_str(nAx)-min_str(nAx))
            zc.zcFlag.6 = 0  ; Not used
            zc.zcFlag.7 = 1  ; Enable end of stroke search
            zc.zcFlag.8 = 0  ; Disable zero cycle ON_POSITION
            zc.zcFlag.9 = 0  ; Disable positioning on preset
elevation
            zc.zcFlag.10 = 0 ; Disable repositioning within software
limit switches.
        endif

        ; Call the move function
        result = mva_zc 486(Flag, zc)
    end_motion
end_rule
```

One-way rotary axis: zero cycle towards the micro + index with preliminary search for the micro variation

Validity: **Rule**

Case 1: the axis starts from the PS point in area A

```
Axis cycle repeat every 360°

                    Direction to high quote (positive speed)
            ------------------------------------------------------------>

 0°        Axis cycle        360°  0°        Axis cycle        360°  0°        Axis cycle        360°
|---------------------------------|---------------------------------|---------------------------------|

                                                CH0
                                                 |
|          |------|          |          |------|          |          |------|          |
|     A    |  B   |     A    |     A    |  B   |     A    |     A    |  B   |     A    |
|----------|      |----------|----------|      |----------|----------|      |----------|
|       ^     ^     ^                     ^  ^
        |     |     |                     |  |
        PS    P1    P2                    |  PF
                                          |
                                      Preset quote

        |-->---|                                  Preliminary mission 1 (+velpos)
               |-->---|                            Preliminary mission 2 (+velpos)
                      |----------------->----------| Zero cycle mission (+velZC)
```

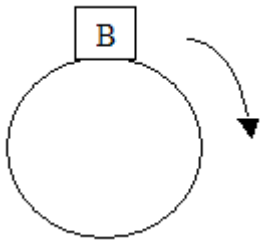
Phases:

- The preliminary search for the micro variation is enabled.
- Preliminary Mission 1: the axis moves at the speed (+*velPos*) until it encounters the first micro front (in this case a rising edge), then decelerates and stops at position **P1**.
- Since the micro is in 1 and cycle0 is completed with micro in 1 an additional preliminary mission is needed.
- Preliminary Mission 2: the axis moves at the speed (+*velPos*) until it encounters the falling edge of the micro, then decelerates and stops at position **P2**.
- Final Mission (CYCLE0): the final movement is performed with a maximum stroke equal to the value of parameter *spSch*, with the set speed (+*velZc*).
  When the first **CH0** encoder with the micro in 1 is found, the mission is stopped and the axis stops in **PF**.
- The value obtained by adding the delta quote (**PF**-quote read on the first **CH0** encoder with the micro in 1) to the *Preset* parameter, is assigned as the new real quote.
- The predefined variable c0_done is updated by setting the axis related bit to 1.

NOTE: The length of cam B read from the micro must be greater

Legend:

- *A, area where the micro reads 0*
- *B, area where the micro reads 1*
- *PS, initial start point for mva_zc 486()*
- *PF, target stop point for mva_zc 486()*
- *P1, intermediate point*
- *P2, intermediate point*
- *CH0, 0 channel of the incremental encoder*



than one encoder revolution to ensure that CH0 is ready with the micro in 1.

Case 2: the axis starts from the PS point in area B

```
Axis cycle repeat every 360°
                        Direction to high quote (positive speed)
              ---------------------------------------------------------->

 0°        Axis cycle        360°  0°        Axis cycle        360°  0°        Axis cycle        360°
|----------------------------------|----------------------------------|----------------------------------|

                                                CH0
|            |------|            |            |------|            |            |------|            |
|     A      |  B   |     A      |     A      |  B   |     A      |     A      |  B   |     A      |
|------------|      |------------|------------|      |------------|------------|      |------------|
|                ^     ^                       ^ ^                                |
                |     |                       | |
               PS    P1                       | PF
                                               |
                                           Preset quote
                |-->--|                                Preliminary mission 1 (+velPos)
                      |-------------->-------------|    Zero cycle mission (+velZC)
```

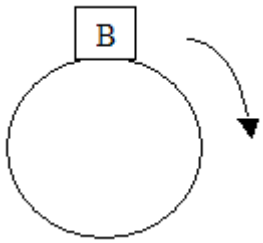
Phases:

- The preliminary search for the micro variation is enabled.
- Preliminary Mission 1: the axis moves at the speed (+*velPos*) until it encounters the first micro front (in this case a falling edge), then decelerates and stops at position **P1**.
- Since the micro is in 0 and cycle0 is completed with micro in 1 no other preliminary missions are required.
- Final Mission (CYCLE0): the final movement is performed with a maximum stroke equal to the value of parameter *spSch*, with the set speed (+*velZc*).
  When the first **CH0** encoder with the micro in 1 is found, the mission is stopped and the axis stops in **PF**.
- The value obtained by adding the delta quote (**PF**-quote read on the first **CH0** encoder with the micro in 1) to the *Preset* parameter, is assigned as the new real quote.
- The predefined variable c0_done is updated by setting the axis related bit to 1.

NOTE: The length of cam B read from the micro must be greater

Legend:

- *A, area where the micro reads 0*
- *B, area where the micro reads 1*
- *PS, initial start point for* mva_zc 486*()*
- *PF, target stop point for* mva_zc 486*()*
- *P1, intermediate point*
- *CH0, 0 channel of the incremental encoder*



than one encoder revolution to ensure that CH0 is ready with the micro in 1.

Examples

Example of use:

```
INT result
STRU_MVZC 863 zc
INT Flag
...
rule 123
    axes 1
    motion
        ; First time initialization
        if (first_time 321())
            Flag = true
            zc.nAx    = 1 ; Axis number
            zc.preset = 0 ; Preselection quotas
            zc.velPos = rVelc0 ; Speed for preliminary movements
mva_zc
            zc.velZC  = rVelc0 * 0.5 ; Speed for final movement
CYCLE0 mva_zc
            zc.acc    = rMaxAcc * 0.5 ; Acceleration to be used in
all movements
            zc.jerk   = rMaxAcc * 10 ; Derivative of acceleration
            zc.spSCh  = 20000 ; Maximum space of the final movement
CYCLE0
            zc.filter = 0 ; Time (sec) of micro signal filtering
            zc.zcFlag.0 = 1  ; Positive speed (since one-way)
            zc.zcFlag.1 = 1  ; Wants micro in 1 when looking for CH0
            zc.zcFlag.2 = 1  ; Enable full-precision
            zc.zcFlag.3 = 0  ; Enabled use of the micro
            zc.zcFlag.4 = 1  ; One-way search
            zc.zcFlag.5 = 0  ; Preliminary unrestricted excursion
missions
            zc.zcFlag.6 = 0  ; Enable search first raising transition
to micro (only for one-way)
            zc.zcFlag.7 = 0  ; Disable end of stroke search (because
micro enabled)
            zc.zcFlag.8 = 0  ; Disabilita ciclo di zero ON_POSITION
            zc.zcFlag.9 = 0  ; Disable positioning on preset
elevation
            zc.zcFlag.10 = 0 ; Disable repositioning within software
limit switches.
        endif

        ; Call move function
        result = mva_zc 486(Flag, zc)
    end_motion
```

```
end_rule
```

One-way rotary axis: zero cycle towards the micro + index without preliminary search for the micro variation

Validity: **Rule**

Case 1: the axis starts from the PS point in area A

```
Axis cycle repeat every 360°

                    Direction to high quote (positive speed)
             ------------------------------------------------------------>

 0°        Axis cycle        360°  0°        Axis cycle        360°  0°        Axis cycle        360°
|---------------------------------|---------------------------------|---------------------------------|

                                                  CH0
                                                   |
|          |------|          |          |------|          |          |------|          |
|     A    |  B   |     A    |     A    |  B   |     A    |     A    |  B   |     A    |
|----------|      |----------|----------|      |----------|----------|      |----------|
|                    ^         |                ^  ^         |                           |
                     |                          |  |
                    PS                          |  PF
                                                |
                                            Preset quote

                    |----------------->-----------|  Zero cycle mission (+velZC)
```

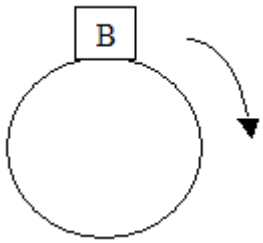
Phases:

- The micro state is read to establish the first movement. Since micro is in 0 and cycle0 is completed with micro in 1, no preliminary missions are needed.
- Final Mission (CYCLE0): the final movement is performed with a maximum stroke equal to the value of parameter *spSch*, with the set speed (+*velZc*). When the first **CH0** encoder with the micro in 1 is found, the mission is stopped and the axis stops in **PF**.
- The value obtained by adding the delta quote (**PF**-quote read on the first **CH0** encoder with the micro in 1) to the *Preset* parameter, is assigned as the new real quote.
- The predefined variable c0_done is updated by setting the axis related bit to 1.

NOTE: The length of cam B read from the micro must be greater than one encoder revolution to ensure that CH0 is ready with the micro in 1.

Legend:

- *A, area where the micro reads 0*
- *B, area where the micro reads 1*
- *PS, initial start point for mva_zc 486()*
- *PF, target stop point for mva_zc 486()*
- *CH0, 0 channel of the incremental encoder*



Case 2: the axis starts from the PS point in area B

```
Axis cycle repeat every 360°
                    Direction to high quote (positive speed)
            ------------------------------------------------------------>

 0°        Axis cycle        360°  0°        Axis cycle        360°  0°        Axis cycle        360°
|-----------------------------------|-----------------------------------|-----------------------------------|

                                                  CH0
                                                   |
|          |------|          |          |------|          |          |------|          |
|    A     |  B   |    A     |    A     |  B   |    A     |    A     |  B   |    A     |
|----------|      |----------|----------|      |----------|----------|      |----------|
|                ^     ^                  ^ ^                |                          |
                 |     |                  | |
                PS    P1                  | PF
                                          |
                                       Preset quote

                 |-->--|                          Preliminary mission 1 (+velPos)
                       |------------->------------| Zero cycle mission (+velZC)
```

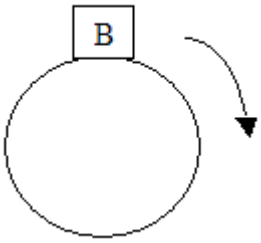
Phases:

- The micro state is read to establish the first movement. Since micro is in 1 and cycle0 is completed with micro in 1, a preliminary mission is required.
- Preliminary Mission 1: the axis moves at the speed (+*velPos*) until it encounters the micro falling edge, then decelerates and stops in **P1**.
- Final Mission (CYCLE0): the final movement is performed with a maximum stroke equal to the value of parameter *spSch*, with the set speed (+*velZc*). When the first **CH0** encoder with the micro in 1 is found, the mission is stopped and the axis stops in **PF**.
- The value obtained by adding the delta quote (**PF**-quote read on the first **CH0** encoder with the micro in 1) to the *Preset* parameter, is assigned as the new real quote.
- The predefined variable c0_done is updated by setting the axis related bit to 1.

NOTE: The length of cam B read from the micro must be greater than one encoder revolution to ensure that CH0 is ready with the micro in 1.

Legend:

- *A, area where the micro reads 0*
- *B, area where the micro reads 1*
- *PS, initial start point for mva_zc 486()*
- *PF, target stop point for mva_zc 486()*
- *P1, intermediate point*
- *CH0, 0 channel of the incremental encoder*



Examples

Example of use:

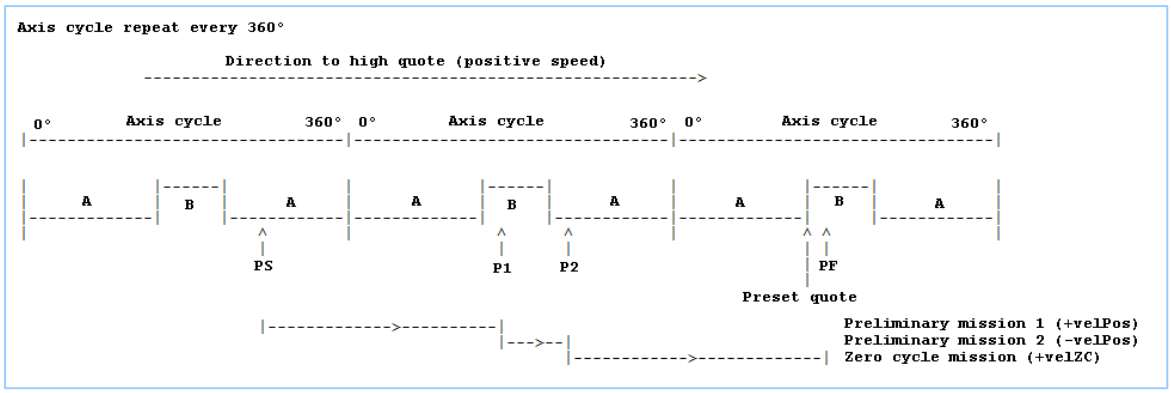
```
INT result
STRU_MVZC 863 zc
INT Flag
...
rule 123
    axes 1
    motion
        ; First time initialization
        if (first_time 321())
            Flag = true
            zc.nAx    = 1 ; Axis number
            zc.preset = 0 ; Preselection quotas
            zc.velPos = rVelc0 ; Speed for preliminary movements
mva_zc 486
            zc.velZC  = rVelc0 * 0.5 ; Speed for final movement
CYCLE0 mva_zc
            zc.acc    = rMaxAcc * 0.5 ; Acceleration to be used in
all movements
            zc.jerk   = rMaxAcc * 10 ; Derivative of acceleration
            zc.spSCh  = 20000 ; Maximum space of the final movement
CYCLE0
            zc.filter = 0 ; Time (sec) of micro signal filtering
            zc.zcFlag.0 = 1 ; Positive speed (since one-way)
            zc.zcFlag.1 = 1 ; Wants micro in 1 when looking for CH0
            zc.zcFlag.2 = 1 ; Enable full-precision
            zc.zcFlag.3 = 0 ; Enable use of micro
            zc.zcFlag.4 = 1 ; One-way search
            zc.zcFlag.5 = 0 ; Preliminary unrestricted excursion
missions
            zc.zcFlag.6 = 1 ; Disable search first transition to
micro up (only for one-way)
            zc.zcFlag.7 = 0 ; Disable paracart search (because micro
enabled)
            zc.zcFlag.8 = 0 ; Disable zero cycle ON_POSITION
            zc.zcFlag.9 = 0 ; Disable positioning on preset elevation
            zc.zcFlag.10 = 0 ; Disable repositioning within software
limit switches.
        endif

        ; Call move function
        esito = mva_zc 486(Flag, zc)
    end_motion
end_rule
```

© 2025 Robox SpA

One-way rotary axis: zero cycle towards the micro with preliminary search for the micro variation

Validity: **Rule**

Case 1: the axis starts from the PS point in area A

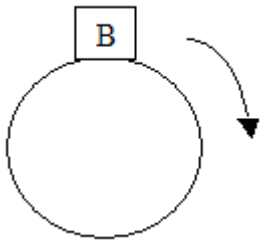


Phases:

- The preliminary search for micro variation is enabled.
- Preliminary Mission 1: the axis moves at speed (+*velPos*) until it encounters the first micro rising edge, then decelerates and stops at position **P1**.
- Since the micro is in 1 and cycle0 is completed with micro in 1, an additional preliminary mission is needed.
- Preliminary mission 2: the axis moves at speed (+*velPos*) until it encounters the falling edge of the micro, then decelerates and stops at position **P2**.
- Final mission (CYCLE0): the final movement is performed with a maximum stroke equal to the value of the *spSch* parameter, with the set speed (+*velZc*). When the rising edge of the micro is found, the mission is stopped and the axis stops in **PF**.
- The value obtained by summing the delta quote (**PF**-quote read at the rising edge of the micro) to the *Preset* parameter, is assigned as new real quote.
- The predefined variable c0_done is updated by setting the axis related bit to 1.

Legend:

- *A, area where the micro reads 0*
- *B, area where the micro reads 1*
- *PS, initial start point for* *mva_zc* 486*()*
- *PF, target stop point for* *mva_zc* 486*()*
- *P1, intermediate point*
- *P2, intermediate point*



Case 2: the axis starts from the PS point in area B

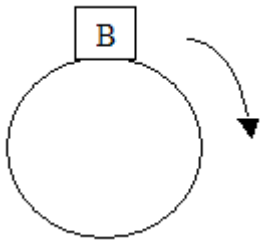
```
Axis cycle repeat every 360°

                    Direction to high quote (positive speed)
              ------------------------------------------------------------>

 0°      Axis cycle          360°  0°      Axis cycle          360°  0°      Axis cycle          360°
|---------------------------------|---------------------------------|---------------------------------|

|             |------|            |             |------|            |             |------|            |
|     A       |  B   |     A      |     A       |  B   |     A      |     A       |  B   |     A      |
|-------------|      |------------|-------------|      |------------|-------------|      |------------|
|                 ^     ^                       ^ ^                  |                                |
                  |     |                       | |
                 PS    P1                       | PF
                                                |
                                          Preset quote
                 |-->--|                                Preliminary mission 1 (+velPos)
                       |------------>------------| Zero cycle mission (+velZC)
```

Phases:

- The preliminary search for micro variation is enabled.
- Preliminary Mission 1: the axis moves at speed (+*velPos*) until it encounters the first micro falling edge, then decelerates and stops at position **P1**.
- Since the micro is in 0 and cycle0 is completed with micro in 1, no preliminary missions are required.
- Final mission (CYCLE0): the final movement is performed with a maximum stroke equal to the value of the *spSch* parameter, with the set speed (+*velZc*). When the rising edge of the micro is found, the mission is stopped and the axis stops in **PF**.
- The value obtained by summing the delta quote (**PF**-quote read at the rising edge of the micro) to the *Preset* parameter, is assigned as new real quote.
- The predefined variable c0_done is updated by setting the axis related bit to 1.

Legend:

- *A, area where the micro reads 0*
- *B, area where the micro reads 1*
- *PS, initial start point for mva_zc 486()*
- *PF, target stop point for mva_zc 486()*
- *P1, intermediate point*



Examples

Example of use:

```
INT result
STRU_MVZC 863 zc
```

```
INT Flag
...
rule 123
    axes 1
    motion
        ; First time initialization
        if (first_time 321())
            Flag = true
            zc.nAx    = 1 ; Axis number
            zc.preset = 0 ; Preselection quotas
            zc.velPos = rVelc0 ; Speed for preliminary movements
mva_zc 486
            zc.velZC  = rVelc0 * 0.5 ; Speed for final movement
CYCLE0 mva_zc 486
            zc.acc    = rMaxAcc * 0.5 ; Acceleration to be used in
all movements
            zc.jerk   = rMaxAcc * 10 ; Derivative of acceleration
            zc.spSCh  = 20000 ; Maximum space of the final movement
CYCLE0
            zc.filter = 0 ; Time (sec) of micro signal filtering
            zc.zcFlag.0 = 1 ; Positive speed (given that one-way)
            zc.zcFlag.1 = 1 ; Selects the micro's rising edge
            zc.zcFlag.2 = 0 ; Disables full-precision
            zc.zcFlag.3 = 0 ; Enables use of the micro
            zc.zcFlag.4 = 1 ; One-way search
            zc.zcFlag.5 = 0 ; Preliminary missions of unrestricted
excursion
            zc.zcFlag.6 = 0 ; Enable search first upward transition
of micro (only for one-way)
            zc.zcFlag.7 = 0 ; Disable paracarro search (because micro
enabled)
            zc.zcFlag.8 = 0 ; Disables zero cycle ON_POSITION
            zc.zcFlag.9 = 0 ; Disable positioning on preset elevation
            zc.zcFlag.10 = 0 ; Disables repositioning within the
software limit switches.
        endif

        ; Call move function
        result = mva_zc 486(Flag, zc)
    end_motion
end_rule
```

One-way rotary axis: zero cycle towards the micro without preliminary search for the micro variation

Validity: **Rule**

Case 1: The axis starts from the PS point in area A

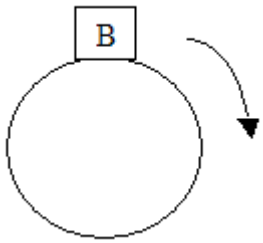
```
Axis cycle repeat every 360°

                    Direction to high quote (positive speed)
            ---------------------------------------------------------->

 0°        Axis cycle        360° 0°        Axis cycle        360° 0°        Axis cycle        360°
|----------------------------------|----------------------------------|----------------------------------|

|            |------|            |            |------|            |            |------|            |
|     A      |  B   |     A      |     A      |  B   |     A      |     A      |  B   |     A      |
|------------|      |------------|------------|      |------------|------------|      |------------|
|                       ^                     |                     ^ ^                     |
                        |                     |                     | |
                       PS                     |                    PF |
                                              |
                                          Preset quote
                      |--------------->--------|  Zero cycle mission (+velZC)
```

Phases:

- The micro state is read to define the first movement.
- Since the micro is in 0 and cycle0 is completed with micro in 1, no additional preliminary missions are required.
- Final mission (CYCLE0): the final movement is performed with a maximum stroke equal to the value of the *spSch* parameter, with the set speed (+*velZc*). When the rising edge of the micro is found, the mission is stopped and the axis stops in **PF**.
- The value obtained by summing the delta quote (**PF**-quote read at the rising edge of the micro) to the *Preset* parameter, is assigned as new real quote.
- The predefined variable c0_done is updated by setting the axis related bit to 1.

Legend:

- *A, area where the micro reads 0*
- *B, area where the micro reads 1*
- *PS, initial start point for* *mva_zc* 486*()*
- *PF, target stop point for* *mva_zc* 486*()*



Case 2: the axis starts from the PS point in area B

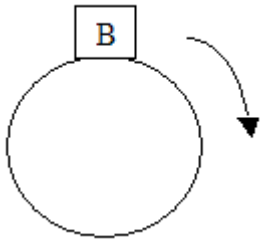
```
Axis cycle repeat every 360°

                    Direction to high quote (positive speed)
            ---------------------------------------------------------->

 0°        Axis cycle        360° 0°        Axis cycle        360° 0°        Axis cycle        360°
|----------------------------------|----------------------------------|----------------------------------|

|            |------|            |            |------|            |            |------|            |
|     A      |  B   |     A      |     A      |  B   |     A      |     A      |  B   |     A      |
|------------|      |------------|------------|      |------------|------------|      |------------|
|                ^     ^                     |                     ^ ^                     |
                 |     |                     |                     | |
                PS    P1                     |                    PF |
                                              |
                                          Preset quote
                 |-->--|                          Preliminary mission (+velPos)
                       |------------>------------|  Zero cycle mission (+velZC)
```

Phases:

- The micro state is read to define the first movement.
- Since the micro is in 1 and cycle0 is completed with micro in 1, a preliminary mission is needed.
- Preliminary Mission 1: the axis moves at speed (+*velPos*) until it encounters the micro falling edge, then it decelerates and stops in **P1**.
- Final mission (CYCLE0): the final movement is performed with a maximum stroke equal to the value of the *spSch* parameter, with the set speed (+*velZc*). When the rising edge of the micro is found, the mission is stopped and the axis stops in **PF**.
- The value obtained by subtracting the delta quote (**PF**-quote read at the rising edge of the micro) to the *Preset* parameter, is assigned as new real quote.
- The predefined variable c0_done is updated by setting the axis related bit to 1.

Legend:

- *A, area where the micro reads 0*
- *B, area where the micro reads 1*
- *PS, initial start point for mva_zc 486()*
- *PF, target stop point for mva_zc 486()*
- *P1, point intermediate*



Examples

Example of use:

```
INT result
STRU_MVZC 863 zc
INT Flag
...
rule 123
    axes 1
    motion
        ; First time initialization
        if (first_time 321())
            Flag = true
            zc.nAx     = 1 ; Axis number
            zc.preset = 0 ; Preselecton quotas
            zc.velPos = rVelc0 ; Speed for preliminary handling
mva_zc
            zc.velZC = rVelc0 * 0.5 ; Speed for final movement CYCLE0
mva_zc
            zc.acc = rMaxAcc * 0.5 ; Acceleration to be used in all
movements
```

© 2025 Robox SpA

```
            zc.jerk = rMaxAcc * 10 ; Derivative of acceleration
            zc.spSCh = 20000 ; Maximum space of the final movement
CYCLE0
            zc.filter = 0 ; Time (sec) for filtering the micro signal
            zc.zcFlag.0 = 1 ; Positive velocity (since one-way)
            zc.zcFlag.1 = 1 ; Selects the rising edge of the micro
            zc.zcFlag.2 = 0 ; Disables full-precision
            zc.zcFlag.3 = 0 ; Enables use of the micro
            zc.zcFlag.4 = 1 ; One-way search
            zc.zcFlag.5 = 0 ; Preliminary missions of unrestricted
excursion
            zc.zcFlag.6 = 1 ; Disable search first upward transition
of micro (only x one-way)
            zc.zcFlag.7 = 0 ; Disable paracarro search (because micro
enabled)
            zc.zcFlag.8 = 0 ; Disable zero cycle ON_POSITION
            zc.zcFlag.9 = 0 ; Disable positioning on preset elevation
            zc.zcFlag.10 = 0 ; Disables repositioning within the
software limit switches.
        endif

        ; Call move function
        result = mva_zc 486(Flag, zc)
    end_motion
end_rule
```

Categories

Mathematical functions/instructions

List of all mathematical/trigonometric functions/instructions available in R3 programming language.

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)

Keyword	R	T	F/I	Description
abs 133()	R	T	F	Absolute value of a quantity
abs_mod 134()	R	T	F	Remainder of division
acos 134()	R	T	F	Arco cosine
asin 155()	R	T	F	Arco sine
atan 156()	R	T	F	Arco tangent
atan2 156()	R	T	F	Arco tangent
cos 265()	R	T	F	Cosine

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
fract 324()	R	T	F	Returns the fractional part of the argument
log 362()	R	T	F	Returns the natural logarithm of the argument
log10 363()	R	T	F	Returns the base10 logarithm of the argument
mod 412()	R	T	F	Returns the remainder of the division between numeric variables
pow 500()	R	T	F	Exponentiation
rand32 531()	R	T	F	Returns a random number (i32)
random 531()	R	T	F	Returns a random number (i16)
random_value 531()	R	T	F	Returns a random number (real)
range 532()	R	T	F	Check if the variable is in the range
range_adj 533()	R	T	F	Scaling of a variable
round 539()	R	T	F	Returns the nearest integer
sign 574()	R	T	F	Returns the sign of the variable
sin 576()	R	T	F	Sine
sqrt 594()	R	T	F	Square root
tan 615()	R	T	F	Tangent
trunc 624()	R	T	F	Returns the integer part of the argument

© 2025 Robox SpA

Safety functions/instructions

List of all safety functions/instructions available in R3 programming language.

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
safe_appl_ack 541()	R	T	F	Sets the value of the Application Acknowledge of a safety island
safe_appl_diag 542()	R	T	F	Returns the status of the Application Diagnostics of a safety island
safe_diag 544()	R	T	F	Returns the diagnostic register of the desired satellite
safe_diag_reset 546()	R	T	F	Sends a Diagnostic Reset to the selected satellite
safe_enab_out 547()	R	T	F	Sets the value of the selected island's output enable register
safe_oper_ack 548()	R	T	F	Sets an Operation Acknowledge on the selected satellite
safe_prj_info 549()	-	T	F	Gets information regarding the safety project running on the specified island
safe_sts 550()	R	T	F	Returns the input/output safety status of the selected satellite

© 2025 Robox SpA

Rule/task functions/instructions

List of all functions/instructions available in R3 programming language for rule and task management.

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
group 694()	R	T	I	Enables RULE (RC) executor
int_after 698()	R	T	I	Enables RULE on a delayed time (priority 1)
int_inp 699()	R	T	I	Enables RULE on a fast input variation (priority 1)
int_timer 701()	R	T	I	Enables RULE at fixed frequency (priority 1)
mt_ats 423()	R	T	F	Builds an atomic semaphore
mt_ats2 424()	R	T	F	Builds an atomic semaphore
mt_en 706()	R	T	I	Enables execution by RTE of the specified time-sharing task(s) (priority 7)
mt_dis 706()	R	T	I	Disables execution by RTE of the specified task(s) in time-sharing (priority 7)
mt_wait 707()	R	T	I	Holds (sleep) the task until the expressed condition becomes true
order 708()	R	T	I	Sets the order of execution (OC) of the RULE executors (RC)
ref_prologue 718()	R	-	I	RULE always active regardless of rule counters (RC)

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
rule 720()	R	-	I	Rule definition
rule_epilogue 728()	R	-	I	Rule always active regardless of rule counters (RC)
rule_freq 729()	R	T	I	Definition of sample interval (SI) of a RULE (priority 2)
rule_init 730()	R	-	I	Initialization rule (runs only at switch-on)
rule_periodic 730()	R	T	I	Enable periodic RULE (priority 4) at fixed frequency
rule_prologue 732()	R	-	I	Rule always active regardless of rule counters (RC)

Alarms management

List of all functions/instructions available in R3 programming language for alarm management.

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
al2am 142()	R	T	F	Returns information about the AM of a specific alarm
alarm_set 151()	R	T	F	Sets an alarm
alarm_set_us 152()	R	T	F	Sets an alarm with USV
alarm_reset 145()	R	T	F	Reset of an alarm
alarm_reset_par 146()	R	T	F	Reset of an alarm specifying parameters

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
alarm_reset_par_us 149()	R	T	F	Reset of an alarm specifying USV parameters
ah_cmd 135()	R	T	F	Commands for alarm history
ah_get_d 136()	R	T	F	Returns the alarm info in the alarm history in a struct
ah_get_s 137()	R	T	F	Returns the alarm info in the alarm history in a string
ah_info 139()	R	T	F	Returns the info of alarms in the alarms history
ah_log 140()	R	T	F	Inserts a comment in the alarms history
ah_log_us 141()	R	T	F	Inserts a comment in the alarms history with USV
as_get_d 154()	R	T	F	Returns the alarm info in the alarm stack in a struct
inf_report 697()	R	T	I	Allows an information to be entered in the system report
is_alarm_present 357()	R	T	F	Checks the presence of the alarm in the stack
is_alarm_par_present 353()	R	T	F	Checks the presence of the alarm and parameter in the stack
is_alarm_par_present_us 356()	R	T	F	Checks the presence of the alarm with USV parameter in the stack

© 2025 Robox SpA

NOTE: Use the keyword sys_log for self-generation in CF of a log file.

NOTE: Use the keyword sys_flag to add information to the system diagnostics obtained through the MREPORT command.

Axioline Management

List of all Axioline functions/instructions available in R3 programming language for managing the Axioline bus.

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
axio_robj 162()	-	T	F	Function to read objects on Axioline fieldbus
axio_wobj 177()	-	T	F	Function to write objects to Axioline fieldbus
axio_pm_r 160()	-	T	F	Function to read process data input from a module AXL_F_PM_EF_1 F
axio_iol_r 157()	-	T	F	Function to read a data area from known master IOlink
axio_iol_w 159()	-	T	F	Function to write a data area to known master IOlink
get_hw_info 326()	-	T	F	Allows to get information about boards directly connected to the CPU

© 2025 Robox SpA

Serial channel management

List of all functions/instructions available i R3 programming language for managing RS232/RS422/RS485 serial channels.

Legend: R = permitted use in Rules - T = permitted use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
ser_485_tx_sts 559 ()	R	T	F	Function that returns the status of the last transmission made on the specified device
ser_bcc_h_init 561()	R	T	F	Initializes the serial port for BCC3 communication between controls
ser_bin_in 562()	R	T	F	Function that allows a specified number of characters to be read from the specified device
ser_bin_out 563()	R	T	F	Sends characters on the serial channel
ser_bin_ready 563()	R	T	F	Reports whether there are characters on the serial channel
ser_buff_free 564()	R	T	F	Number of free bytes on the TX buffer
ser_buff_queue 564 ()	R	T	F	Number of occupied bytes on the TX buffer
ser_con 737()	R	T	I	Sets the parameters of the serial channel
ser_flush 564()	R	T	F	Cleans all data in the TX and RX buffers of the specified serial channel
ser_in 565()	R	T	F	Receives the terminated string

© 2025 Robox SpA

Legend: R = permitted use in Rules - T = permitted use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
				from serial channel
ser_mod 566()	R	T	F	Sets the working mode of the specified serial channel
ser_out 567()	R	T	F	Sends the terminated string to the serial channel
ser_r_sts 567()	R	T	F	Reads the status word from the serial channel
ser_ready 568()	R	T	F	Reports the presence of a terminated string on the serial channel
ser_w_sts 738()	R	T	I	Sets the status word of the serial channel

CANopen/EtherCAT Management

List of all functions/instructions available in R3 programming language for managing the CANopen DS301/DS401/DS402 protocol.

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
co_send_sdo 242()	R	T	F	Send an SDO (service data object) message.
co_asw_sdo 215()	R	T	F	Returns the response to the previous instruction co_send_sdo
co_controlword_om 218()	R	T	F	Function to write some bits of the controlword
co_controlword_state 220()	R	T	F	Function to effect transitions in the

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
				DS 402 state machine
co_statusword 246 ()	R	T	F	Returns the contents of the status word of the indicated workstation (WS)
co_par_download 224()	-	T	F	Function to load parameters to the drive, by reading them from Compact Flash (CF) control
co_par_upload 227 ()	-	T	F	Function to save to a file in the CF the parameters, by reading them from drive
co_get_ws_diagno 221()	R	T	F	Function to read status word information of the indicated WS (CANopen only)
co_rerr 231()	R	T	F	Returns error information of the indicated WS
co_rnmt 232()	R	T	F	Function to returns the NMT operating mode of the indicated WS
co_wnmt 248()	R	T	F	Imposes the NMT status of the indicated WS
co_robj 234()	R	T	F	Function to read fieldbus objects
co_wobj 249()	R	T	F	Function to write fieldbus objects
co_rpdo 241()	R	T	F	Reads a CANopen object via PDO (CANopen only)

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
co_sync 247()	R	T	F	Function to enable or disable transmission of the SYNC message
co_wpdo 254()	R	T	F	Writes a CANopen object via PDO (CANopen only)
coe_robj 255()	R	T	F	Function to read COE fieldbus objects
coe_wobj 259()	R	T	F	Function to write COE fieldbus objects
ecat_dev_on_off 284()	-	T	F	Function to enable/disable a device on EtherCAT line
ecat_get_state 287()	R	T	F	Function to read the status of the EtherCAT line
ecat_probe 288()	-	T	F	Function to read the information of a device on EtherCAT line
ecat_set_state 290()	R	T	F	Function to set the NMT state of the EtherCAT line
soe_ridn 577()	R	T	F	Function to read SOE fieldbus objects
soe_widn 589()	R	T	F	Function to write SOE fieldbus objects

© 2025 Robox SpA

CNET Management

List of all functions/instructions available in R3 programming language for handling the CNET protocol.

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
cnet_diag()	R	T	F	CNET management function. Reserved
cnet_get_time_sync()	R	T	F	CNET management function. Reserved
cnet_recive()	R	T	F	CNET management function. Reserved
cnet_receive_status()	R	T	F	CNET management function. Reserved
cnet_rq_sync()	R	T	F	CNET management function. Reserved
cnet_send()	R	T	F	CNET management function. Reserved
cnet_status()	R	T	F	CNET management function. Reserved

BCC Communication Management

List of all functions/instructions available in R3 programming language to handle a communication with the BCC protocol.

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
bcc_h_setup 202()	R	T	F	Prepares the data structure for responding to a message

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
bcc_receive 204()	R	T	F	Reads a BCC3 message
bcc_send 206()	R	T	F	Sends a BCC3 message
ser_bcc_h_init 561()	-	T	F	Initializes a serial port to communicate with the BCC3 protocol
bcc_get_double 197()	R	T	F	Reads a double from a BCC_MSGUSER structure
bcc_get_float 197()	R	T	F	Reads a float from a BCC_MSGUSER structure
bcc_get_i16 198()	R	T	F	Reads an i16 from a BCC_MSGUSER structure
bcc_get_i32 198()	R	T	F	Reads an i32 from a BCC_MSGUSER structure
bcc_get_i8 199()	R	T	F	Reads an i8 from a BCC_MSGUSER structure
bcc_get_string 199()	R	T	F	Reads a string from a BCC_MSGUSER structure
bcc_get_tcp_clients_info 200()	R	T	F	Reads an i32 from a STRU_BCC_TCP_CLIENT structure
bcc_get_u16 201()	R	T	F	Reads an u16 from a BCC_MSGUSER structure
bcc_get_u32 201()	R	T	F	Reads an u32 from a BCC_MSGUSER structure

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
bcc_get_u8 202()	R	T	F	Reads an u8 from a BCC_MSGUSER structure
bcc_reply_ack 204()	R	T	F	Sends a positive response (ACK) to a received message
bcc_reply_msg 205()	R	T	F	Sends a response to a received message
bcc_reply_nack 206()	R	T	F	Sends a negative (NACK) response to a received message
bcc_set_double 656()	R	T	I	Writes a double to a BCC_MSGUSER structure
bcc_set_float 657()	R	T	I	Writes a float into a BCC_MSGUSER structure
bcc_set_i16 658()	R	T	I	Writes an i16 into a BCC_MSGUSER structure
bcc_set_i32 658()	R	T	I	Writes an i32 in a BCC_MSGUSER structure
bcc_set_i8 659()	R	T	I	Writes an i8 into a BCC_MSGUSER structure
bcc_set_string 659()	R	T	I	Writes a string in a BCC_MSGUSER structure
bcc_set_u16 660()	R	T	I	Writes an u16 into a BCC_MSGUSER structure
bcc_set_u32 660()	R	T	I	Writes an u32 into a

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
				BCC_MSGUSER structure
bcc_set_u8 661()	R	T	I	Writes an u8 into a BCC_MSGUSER structure

Flow control management

List of all functions/instructions available in R3 programming language to manage flow control in the program.

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
break 667	R	T	I	Exit from a selection block
call 667	R	T	I	Calling a subprogram
case 669	R	T	I	Variable select construct
cond_val 265	R	T	I	Simplified if-then-else construct
continue 670	R	T	I	For-while construct
default 675	R	T	I	Default variable select construct
do 676	R	T	I	Loop do
dwell 677	-	T	I	Stops the task for *n* seconds
_else 651	R	T	I	Alternative choice block _if
else 677	R	T	I	Alternative choice block if
first_time 321()	R	-	F	Runs only the first time the RULE 720 is activated
for 686	R	T	I	Loop for
function 688	R	T	I	Function (subroutine)

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
goto 693	R	T	I	Jump to label instruction
_if 651	R	T	I	If construct with only one subsequent line
if 696	R	T	I	If construct
__main_loop__ 650	R	T	I	Task loop
restart_point 719	-	T	I	Task execution breakpoint before live_edit 903
select 736	R	T	I	Select construct
while 760	R	T	I	While construct

Endianess conversion management

List of all functions/instructions available in R3 programming language for handling endianess conversion.

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
Reading functions:				
get_i8 329()	R	T	F	Read a byte (I8) from a buffer at a specified offset
get_u8 329()	R	T	F	Read a byte (U8) from a buffer at a specified offset
get_i16 328()	R	T	F	Read a word (I16) from a buffer at a specified offset
get_u16 331()	R	T	F	Read a word (U16) from a buffer at a specified offset
get_i32 328()	R	T	F	Read a dword (I32) from a buffer at a specified offset

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
get_u32 332()	R	T	F	Read a dword (U32) from a buffer at a specified offset
get_i64 692()	R	T	I	Read a qword (I64) from a buffer at a specified offset
get_u64 693()	R	T	I	Read a qword (U64) from a buffer at a specified offset
get_float 325()	R	T	F	Read a float (x32) from a buffer at a specified offset
get_real 330()	R	T	F	Read a real (x64) from a buffer at a specified offset
Writing functions:				
set_i8 743()	R	T	I	Write a byte (I8) from a buffer at a specified offset
set_u8 747()	R	T	I	Write a byte (U8) from a buffer at a specified offset
set_i16 740()	R	T	I	Write a word (I16) from a buffer at a specified offset
set_u16 745()	R	T	I	Write a word (U16) from a buffer at a specified offset
set_i32 741()	R	T	I	Write a dword (I32) from a buffer at a specified offset
set_u32 746()	R	T	I	Write a dword (U32) from a buffer at a specified offset
set_i64 742()	R	T	I	Write a qword (I64) from a

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
				buffer at a specified offset
set_u64 747()	R	T	I	Write a qword (U64) from a buffer at a specified offset
set_float 739()	R	T	I	Write a float (x32) from a buffer at a specified offset
set_real 744()	R	T	I	Write a real (x64) from a buffer at a specified offset
Functions of endian x16 conversions:				
wbe2le 647()	R	T	F	Convert big endian word (x16 or x32) to little endian word (x16)
wle2be 648()	R	T	F	Convert little endian word (x16 or x32) to big endian word (x16)
Endian x32 conversion functions:				
dwbe2le 277()	R	T	F	Convert big endian dword (x32) to little endian dword (x32)
dwle2be 279()	R	T	F	Convert little endian dword (x32) to big endian wdord (x32)
dwbe2lef 278()	R	T	F	Convert big endian dword (x32) to little endian float (x32)
dwle2bef 280()	R	T	F	Convert little endian float (4 bytes) to big

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
				endian dword (U32)
Middle endian x32 conversion functions:				
dwle2mbe 280()	R	T	F	Convert little endian dword (x32) to middle big endian dword (x32)
dwle2mbef 281()	R	T	F	Convert little endian float (4 bytes) to middle big endian dword (U32)
dwmbe2le 283()	R	T	F	Convert middle big endian dword (x32) to little endian dword (x32)
dwmbe2lef 283()	R	T	F	Convert middle big endian (U32) to little endian float (4 bytes)
dwle2mle 281()	R	T	F	Convert little endian dword (x32) to middle little endian dword (x32)
dwle2mlef 282()	R	T	F	Convert little endian float (4 bytes) to middle little endian dword (U32)
dwbe2mbe 278()	R	T	F	Convert big endian dword (x32) to middle big endian dword (x32)
dwbe2mle 279()	R	T	F	Convert big endian dword (x32) to middle little endian dword (x32)
dwmbe2be 282()	R	T	F	Convert middle big endian dword (x32) to

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
				big endian dword (x32)
dwmle2be 284()	R	T	F	Convert middle little endian dword (x32) to big endian dword (x32)
Endian x64 conversion functions, with use of stru_large_int and stru_large_uint:				
qwbe2le 711()	R	T	I	Convert big endian qword (x64) to little endian qword (x64)
qwbe2lef 521()	R	T	F	Convert big endian qword (x64) to little endian real (x64)
qwle2be 712()	R	T	I	Convert little endian qword (x64) to big endian qword (x64)
qwle2bef 714()	R	T	I	Convert little endian real (x64) to big endian qword (x64)
Middle endian x64 conversion functions, with use of stru_large_int and stru_large_uint:				
qwmbe2le 716()	R	T	I	Convert middle big endian qword (x64) to little endian qword (x64)
qwmbe2lef 522()	R	T	F	Convert middle big endian qword (x64) to little endian real (x64)
qwle2mbe 714()	R	T	I	Convert little endian qword (x64) to middle big endian qword (x64)
qwle2mbef 716()	R	T	I	Convert little endian real (x64) to middle big

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
				endian qword (x64)
Dword exchange endian x64 conversion functions, with use of stru_large_int and stru_large_uint:				
qwbedwx2le 711()	R	T	I	Convert dword exchange big endian qword (x64) to little endian qword (x64)
qwbedwx2lef 521()	R	T	F	Convert dword exchange big endian qword (x64) to little endian real (x64)
qwle2bedwx 712()	R	T	I	Convert little endian qword (x64) to dword exchange big endian qword (x64)
qwle2bedwxf 713()	R	T	I	Convert little endian real (x64) to dword exchange big endian real (x64)
Dword exchange middle endian x64 conversion functions, with use of stru_large_int and stru_large_uint:				
qwmbedwx2le 717()	R	T	I	Convert dword exchange middle big endian qword (x64) to little endian qword (x64)
qwmbedwx2lef 522()	R	T	F	Convert dword exchange middle big endian qword (x64) to little endian real (x64)
qwle2mbedwx 712()	R	T	I	Convert little endian qword (x64) to dword exchange middle big endian qword (x64)

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
qwle2mbedwxf 713 ()	R	T	I	Convert little endian real (x64) to dword exchange middle big endian qword (x64)

Counter management

List of all functions/instructions available in R3 programming language for counter management.

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
ct_cd 670()	R	T	I	Counter down
ct_cdr 671()	R	T	I	Counter down ring
ct_cu 672()	R	T	I	Counter up
ct_cur 673()	R	T	I	Counter up ring
ct_cv 269() / ct_cv 674()	R	T	F/I	Counter current value
ct_load 674()	R	T	I	Load PV into the counter CV
ct_pv 270() / ct_pv 675()	R	T	F/I	Counter preset value
ct_qd 270()	R	T	F	Counter down result
ct_qu 271()	R	T	F	Counter up result
ct_reset 675()	R	T	I	Counter CV reset

© 2025 Robox SpA

Data Management

List of all functions/instructions available in R3 programming language for data manipulation.

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
b0 191() / b0 654()	R	T	F/I	Function that returns/writes byte 0 of the argument
b1 192() / b1 654()	R	T	F/I	Function that returns/writes byte 1 of the argument
b2 193() / b2 655()	R	T	F/I	Function that returns/writes byte 2 of the argument
b3 194() / b3 655()	R	T	F/I	Function that returns/writes byte 3 of the argument
b4 194() / b4 655()	R	T	F/I	Function that returns/writes byte 4 of the argument
b5 195() / b5 655()	R	T	F/I	Function that returns/writes byte 5 of the argument
b6 195() / b6 656()	R	T	F/I	Function that returns/writes byte 6 of the argument
b7 196() / b7 656()	R	T	F/I	Function that returns/writes byte 7 of the argument
crc16_be 266()	R	T	F	Function for calculating the CRC (big endian)
crc16_le 268()	R	T	F	Function for calculating the CRC (little endian)
dw0 275() / dw0 676()	R	T	F/I	Function that returns/writes

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
				the dword 0 of the argument
dw1 277() / dw1 677()	R	T	F/I	Function that returns/writes the dword 1 of the argument
fill_mem 685()	R	T	I	Allows to initialize a memory area
move_byte 704()	R	T	I	Copies a memory area (n bytes)
move_mem 705()	R	T	I	Copies a memory area (portion length)
r2m16 527()	R	T	F	Real to U16 conversion
r2m32 528()	R	T	F	Real to U32 conversion
r2m8 529()	R	T	F	Real to U8 conversion
r_ror 524()	R	T	F	Rotation to the right of the indicated number of bits
r_rol 523()	R	T	F	Rotation to the left of the indicated number of bits
r_shl 525()	R	T	F	Arithmetic shift to the left of the indicated number of bits
r_shr 526()	R	T	F	Arithmetic shift to the right of the indicated number of bits
rbit 534()	R	T	F	Bit reset in an integer variable
sb0 551() / sb0 733()	R	T	F/I	Function that returns/writes the argument byte 0 with sign extension

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
sb1 552() / sb1 733()	R	T	F/I	Function that returns/writes byte 1 of the argument with sign extension
sb2 553() / sb2 733()	R	T	F/I	Function that returns/writes byte 2 of the argument with sign extension
sb3 553() / sb3 734()	R	T	F/I	Function that returns/writes byte 3 of the argument with the sign extension
sb4 554() / sb4 734()	R	T	F/I	Function that returns/writes byte 4 of the argument with the sign extension
sb5 554() / sb5 734()	R	T	F/I	Function that returns/writes byte 5 of the argument with the sign extension
sb6 555() / sb6 735()	R	T	F/I	Function that returns/writes byte 6 of the argument with the sign extension
sb7 556() / sb7 735()	R	T	F/I	Function that returns/writes byte 7 of the argument with the sign extension
sbit 556()	R	T	F	Set of bits in an integer variable
sdw0 557() / sdw0 735()	R	T	F/I	Function that returns/writes the dword 0 of the argument with the sign extension

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
sdw1 558() / sdw1 736()	R	T	F/I	Function that returns/writes the argument's dword 1 with the sign extension
shift_left 570()	R	T	F	Shift bits to the left of the indicated memory area
shift_right 571()	R	T	F	Shift bits to the right of the indicated memory area
shl 573()	R	T	F	Logical shift to the left
shr 573()	R	T	F	Logical shift to the right
sizeof 576()	R	T	F	Returns the size of the variable
sum8_mem 611()	R	T	F	Returns the sum of n elements of an array (u8)
sum16_mem 611()	R	T	F	Returns the sum of n elements of an array (u16)
sum32_mem 612()	R	T	F	Returns the sum of n elements of an array (u32)
sumr_mem 612()	R	T	F	Returns the sum of n elements of an array (real)
sw0 612() / sw0 749()	R	T	F/I	Function that returns/writes the argument word 0 with the sign extension
sw1 613() / sw1 749()	R	T	F/I	Function that returns/writes the word 1 of the argument with the sign extension
sw2 614() / sw2 750()	R	T	F/I	Function that returns/writes

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
				the word 2 of the argument with the sign extension
sw3 614() / sw3 750()	R	T	F/I	Function that returns/writes word 3 of the argument with the sign extension
tobcd 619()	R	T	F	Converts the value from binary to BCD (Binary-Coded Decimal)
tobin 620()	R	T	F	Converts the value from BCD to binary
todeg 621()	R	T	F	Converts the value from radians to degrees
torad 621()	R	T	F	Converts the value from degrees to radians
w0 644() / w0 759()	R	T	F/I	Function that returns/writes the argument word 0
w1 645() / w1 759()	R	T	F/I	Function that returns/writes the word 1 of the argument
w2 645() / w2 760()	R	T	F/I	Function that returns/writes word 2 of the argument
w3 646() / w3 760()	R	T	F/I	Function that returns/writes word 3 of the argument
wbit 647()	R	T	F	Set/Reset of bits in an integer variable

© 2025 Robox SpA

NOTE: If you are looking for functions/instructions to read/write data into a buffer see endianess conversion management 103.

Flash management

List of all functions/instructions available in R3 programming language for flash management (independent of hardware type).

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
f_avail 293()	-	T	F	Memory available on Compact Flash (CF)
f_close 294()	-	T	F	Closing a file opened in CF
f_copy 295()	-	T	F	Copying the contents of a file to a new file
f_del 296()	-	T	F	Deleting a file on the CF
f_dir 297()	-	T	F	Information about files contained in CF
f_gets 300()	-	T	F	Reads from CF a file opened in text mode
f_mkdir 301()	-	T	F	Creates a new directory in CF
f_open 302()	-	T	F	Opens a file in CF
f_puts 304()	-	T	F	Writes to a file on the CF that is open in text mode
f_read 305()	-	T	F	Reads from CF a file opened in binary mode
f_read_table_s 308()	-	T	F	Reads from CF an ascii file and loads the values in it into variables (checks for overflow from the buffer)
f_read_table 306()	-	T	F	Reads from CF an ascii file and

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
				loads the values in it into variables
f_rename 310()	-	T	F	Renames a file in CF
f_seek 311()	-	T	F	Moves the pointer inside a file on the CF
f_write 312()	-	T	F	Writes to a file on the CF opened in binary mode
f_format 298()	-	T	F	Formats CF partitions

I/O forcing management

List of all functions/instructions available in R3 programming language for I/O forcing.

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
force 322()	R	T	F	Function to force the state of IO
release 534()	R	T	F	Function to release the forcing of IO
is_forced 358()	R	T	F	Function to control the forcing of IO

Modbus/TCP client management

List of all functions/instructions available in R3 programming language for managing the Modbus/TCP client protocol.

Legend: FC = Function Code - R = Permitted use in Rules - T = Permitted use in Tasks - I = Instruction - F = Function (thus has a return value)					
Keyword	FC	R	T	F/I	Description
mbc_connect 365()	-	-	T	F	Connect client to the

© 2025 Robox SpA

Legend: FC = Function Code - R = Permitted use in Rules - T = Permitted use in Tasks - I = Instruction - F = Function (thus has a return value)					
Keyword	FC	R	T	F/I	Description
					Modbus/TCP server
mbc_init 366()	-	-	T	F	Initialize Modbus/TCP client
mbc_mask_write_reg 368()	22	-	T	F	Write registers with mask
mbc_mask_write_reg_s 370()	22	-	T	F	Write registers with mask via support structure
mbc_read_coils 372()	01	-	T	F	Read coils
mbc_read_coils_s 373()	01	-	T	F	Read coils with support structure
mbc_read_discr_inp 374()	02	-	T	F	Read discrete inputs
mbc_read_discr_inp_s 376()	02	-	T	F	Read discrete inputs with support structure
mbc_read_hold_reg 377()	03	-	T	F	Read holding registers
mbc_read_hold_reg_s 379()	03	-	T	F	Read holding registers with support structure
mbc_read_inp_reg 381()	04	-	T	F	Read input registers
mbc_read_inp_reg_s 382()	04	-	T	F	Read input registers with support structure
mbc_shutdown 384()	-	-	T	F	Close the connection with the Modbus/TCP server
mbc_write_coils 384()	05 (single coil)	-	T	F	Write coils

© 2025 Robox SpA

Legend: FC = Function Code - R = Permitted use in Rules - T = Permitted use in Tasks - I = Instruction - F = Function (thus has a return value)					
Keyword	FC	R	T	F/I	Description
	15 (coils > 1)				
mbc_write_coils_s 386()	05 (single coil) 15 (coils > 1)	-	T	F	Write coils using support structure
mbc_write_multiple_coils 387()	15	-	T	F	Write coils using Modbus function code of multiple corresponding write functions
mbc_write_multiple_coils_s 388()	15	-	T	F	Write coils via support structure using Modbus function code of multiple corresponding write functions
mbc_write_multiple_regs 390()	16	-	T	F	Write registers using Modbus function code of multiple corresponding write functions
mbc_write_multiple_regs_s 392()	16	-	T	F	Write registers via support structure using Modbus function code of multiple corresponding write functions
mbc_write_reg 393()	06 (single reg) 16 (reg > 1)	-	T	F	Write registers
mbc_write_reg_s 395()	06 (single reg) 16 (reg > 1)	-	T	F	Write registers via support structure
mbcu_mask_write_reg 396()	22	-	T	F	Write registers with

© 2025 Robox SpA

Legend: FC = Function Code - R = Permitted use in Rules - T = Permitted use in Tasks - I = Instruction - F = Function (thus has a return value)					
Keyword	FC	R	T	F/I	Description
					mask by specifying the unitID
mbcu_read_coils 398()	01	-	T	F	Read coils by specifying the unitID
mbcu_read_discr_inp 400()	02	-	T	F	Read discrete inputs specifying the unitID
mbcu_read_hold_reg 401()	03	-	T	F	Read holding registers specifying the unitID
mbcu_read_inp_reg 403()	04	-	T	F	Read input registers specifying the unitID
mbcu_write_coils 405()	05 (single coil) 15 (coils > 1)	-	T	F	Write coils by specifying the unitID
mbcu_write_multiple_coils 406()	15	-	T	F	Write coils specifying the unitID using Modbus function code of the corresponding multiple write functions
mbcu_write_multiple_regs 408()	16	-	T	F	Write registers specifying the unitID using Modbus function code of the corresponding multiple write functions
mbcu_write_reg 410()	06 (single reg) 16 (reg > 1)	-	T	F	Write registers by specifying the unitID

© 2025 Robox SpA

NOTE: The Function Code (FC) is a specific code used in a Modbus request to tell the Modbus slave device what type of memory (HOLDING REGISTERS, INPUT COILS, ...) to access and what action to perform on that memory (read or write).

MOT management

List of all functions/instructions for handling MOT (Multiple Output Table) available in R3 programming language.

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
mot_append 413()	R	T	F	Adding a new point to the MOT
mot_clear 417()	R	T	F	Removing all points from the MOT
mot_create 417()	R	T	F	Creating a new MOT
mot_delete 421()	R	T	F	Deleting the MOT
mot_get 421()	R	T	F	Reading data about a point in the MOT
mot_remove 423()	R	T	F	Removing one or more points from the MOT
mv_mot_exec 445()	R	T	F	Executing the MOT

Motion management

List of all functions/instructions available in R3 programming language for axis movement.

NOTE: Refer to the Robox Path Executor (RPE) library manual if you are using an axes group.

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
mv_cam 425()	R	-	F	Performs cubic cam for sections a, b, c
mv_crimper 427()	R	-	F	Crimper function for flowpack
mv_follow 428()	R	-	F	Cut fly.

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
				follow2 is recommended
mv_follow2 432()	R	-	T	Cut fly (master and slave not associated with an axis)
mv_mot_exec 445()	R	T	F	Interpolation function using MOT (Multiple Output Table)table
mv_phase_adj 447()	R	-	F	Phase correction between two axes
mv_phase_adj2 448()	R	-	F	Phase correction between two axes
mv_ramp 451()	R	T	F	Deprecated. Use ramp 530()
mv_reach_target 451()	R	-	F	Docking between two axes
mv_sine_cam 453()	R	-	F	Running a cam (y as a function of x)
mv_synchro 454()	R	-	F	Docking between two axes
mv_table 455()	R	-	F	Performs a Cam for points
mv_tracking 466()	R	-	F	Deprecated. Use tracking 622()
mv_to 457()	R	-	F	Motion toward a target
mv_to_vel 464()	R	-	F	Movement toward a target with a specified final velocity
mv_to_cj 461()	R	-	F	Movement toward a target with controlled jerk (parameters acquired on start only)
mv_to_cjv 461()	R	-	F	Movement toward a target with controlled jerk

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
				(parameters modifiable run-time)
mv_to_cjv_info 463 ()	R	T	F	Auxiliary function to mv_to_cjv()
mva_follow2 442()	R	-	F	Cut fly (only slave associated with an axis)
mva_follow2_m 443 ()	R	-	F	Cut fly (master and slave associated with an axis)
mva_jog 466()	R	-	F	JOG movement (manual)
mva_jog2 468()	R	-	F	JOG movement (manual) with controlled jerk
mva_open_loop 470()	R	-	F	Open loop (missing power RULE)
mva_to_n 471()	R	-	F	Movement toward a target with multiple axes
mva_to_n_cj 475()	R	-	F	Movement toward a target with multiple axes
mva_to_n_v 479()	R	-	F	Movement toward a target with multiple axes (also already in motion)
mva_zc 486()	R	-	F	Homing movement
kin_conv 361()	R	T	F	Calculates acceleration and jerk values
set_kinematics 569 ()	R	-	F	Imposes kinematic values simultaneously (ip, iv, ia)
tracking 622()	R	T	F	Brings the controlled quote toward the specified target

© 2025 Robox SpA

Power set management

List of all functions/instructions available in R3 programming language for interaction with the Power set.

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
exl_lam 292()	R	T	F	Shows the included/excluded logical axes of a Power set
excl_lam 684	R	T	I	Includes/excludes the logical axes of a Power set
excl_pam 293()	R	T	F	Shows the included/excluded physical axes of a Power set
excl_pam 685	R	T	I	Includes/excludes the physical axes of a Power set
ps_exclude_lax 511()	R	T	F	Excludes the logical axes of a Power set
ps_exclude_pax 513()	R	T	F	Excludes the physical axes of a Power set
ps_reinclude_lax 515()	R	T	F	Includes the logical axes of a Power set
ps_reinclude_pax(516()	R	T	F	Includes the physical axes of a Power set
ps_power_enable 514()	R	T	F	Enables the power of the Power set
ps_channel_enable 501()	R	T	F	Enables the axes of the Power set
ps_channel_reset_fault 507()	R	T	F	Sends an alarm reset command
ps_drive_reset 509()	R	T	F	Sends a drive reset command
ps_status 517()	R	T	F	Returns information on the status of the Power set

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
ps_channel_enabled 504()	R	T	F	Returns information about the enabled axes of a Power set
ps_channel_fault 505()	R	T	F	Returns information about the faulted axes of a Power set

String management

List of all functions/instructions available in R3 programming language for ASCII string manipulation.

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
ipaddr_to_str 353()	R	T	F	Converts an IP address from integer to string
str_asc 598()	R	T	F	Converts a character to a number
str_chr 598()	R	T	F	Converts a number to a character
str_compact 599()	R	T	F	Compacts the spaces
str_equal 600()	R	T	F	Compares two strings
str_find 600()	R	T	F	Searches within a string
str_format 601()	R	T	F	Formats the string (sprintf)
str_i 602()	R	T	F	Converts an integer numeric value to ASCII
str_insert 603()	R	T	F	Inserts within a string

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
str_len 604()	R	T	F	Calculates the length of the string
str_mid 605()	R	T	F	Takes a portion of a string
str_r 606()	R	T	F	Converts a real numeric value to ASCII
str_replace 607()	R	T	F	Substitutes within a string
str_val 608()	R	T	F	Returns, in numeric format, a value contained in a string.
str_to_ipaddr 608()	R	T	F	Converts a string containing the IP to a numeric value
tolower 621()	R	T	F	Converts a character to lower case
toupper 622()	R	T	F	Converts a character to uppercase

Timer management

List of all functions/instructions available in R3 programming language for managing timers.

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
tm_on 754()	R	T	I	Timer ON
tm_of 753()	R	T	I	Timer OFF
tm_tp 758()	R	T	I	Pulse timer
tm_sqw 756()	R	T	I	Square wave timer
tm_int 751()	R	T	I	Integral timer
Interaction functions:				

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
tm_et 616() / tm_et 751()	R	T	F/I	Reads/sets the timer time
tm_pt 617() / tm_pt 756()	R	T	F/I	Reads/sets the timer preset
tm_pt2 618() / tm_pt2 756()	R	T	F/I	Reads/sets the second timer preset (timer square)
tm_q 619()	R	T	F	Timer output (coil)
tm_reset 756()	R	T	I	Resets timer time (integral timer)

UDP/IP management

List of all functions/instructions available in R3 programming language for handling the UDP/IP protocol.

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
ipaddr_to_str 353()	R	T	F	Converts IP address from integer to string
str_to_ipaddr 608()	R	T	F	Converts IP address from string to integer
udp_open_client 626()	R	T	F	Opens an UDP client connection
udp_open_server 628()	R	T	F	Opens an UDP server connection
udp_close 625()	R	T	F	Closes an UDP connection
udp_send 636()	R	T	F	Sends an UDP data packet
udp_send_to 640()	R	T	F	Sends an UDP data packet to...
udp_recv 631()	R	T	F	Receives an UDP data packet

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
udp_recv_from 632 ()	R	T	F	Receives an UDP data packet from...
udp_send_notify 638()	R	T	F	Sends an UDP data packet with notify
udp_recv_notify 634 ()	R	T	F	Receives an UDP data packet with notify

Global variable management

List of all functions/instructions available in R3 programming language for handling global variables.

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
gv_create 334()	-	T	F	Creates a new global variable (GV). Creates the information in memory and, if necessary, in retentive memory
gv_destroy 338()	-	T	F	Removes a GV. Deletes the configuration file and memory space
gv_link 342()	R	T	F	Links to an existing GV. Creates an interface to a GV
gv_unlink 347	R	T	F	Removes the interface link to a GV
gv_read 345()	R	T	F	Updates the data of the associated local variable via the interface,

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
				reading from the GV
gv_read_partial 345 ()	R	T	F	Partially updates the data of the associated local variable via the interface, reading from the GV
gv_write 349()	R	T	F	Updates the value of the GV with the value of the associated local variable via the interface
gv_write_partial 349 ()	R	T	F	Partially updates the value of the GV with the value of the associated local variable via the interface
gv_lock 344()	R	T	F	Locks GV access - access is not allowed by any other interface
gv_unlock 348()	R	T	F	Unlocks GV access - access is then allowed by all interfaces
gv_exist 338()	R	T	F	Checks whether the specified string matches a defined GV
gv_name 344()	R	T	F	Gets the name of a GV
gv_datatype 336()	R	T	F	Gets the type of a GV
gv_datatype_byname 337()	R	T	F	Gets the type of a GV giving the name
gv_flags 339()	R	T	F	Gets the flag associated with a GV

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
gv_flags_byname 341()	R	T	F	Gets the flag associated with a GV giving the name
gv_size 346()	R	T	F	Gets the size of a GV
gv_size_byname 347()	R	T	F	Gets the size of a GV giving the name
gv_update_file 348()	-	T	F	Updates the information in the configuration file

Miscellaneous

List of all miscellaneous functions/instructions available in R3 programming language.

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
check_mv_follow2 207()	R	T	F	Checks if the mv_follow2 432() function can be executed
check_mva_follow2 210()	R	T	F	Checks if the mva_follow2 442() function can be executed
check_mva_follow2_m 213()	R	T	F	Checks if the mva_follow2_m 443() function can be executed
command 264()	R	T	F	Executes a command (directive)
dec 271()	R	T	F	Decreases a variable by a specified quantity
delta_enc16 272()	R	T	F	Calculates the difference between two 16-bit units and

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
				handles the roll-over
delta_enc32 273()	R	T	F	Calculates the difference between two 32-bit units and handles the roll-over
diff 274()	R	T	F	Returns true if the value of the variable has changed
fall 314()	R	T	F	Captures the falling edge of a condition
fall_filt 314()	R	T	F	Captures the falling edge of a condition filtered
filter_lp 317()	R	-	F	Low-pass filter on the variable
filter_lp_st 317()	R	-	F	Low-pass filter on the variable that can be used in a for loop
filter_notch 319()	R	-	F	Notch filter on the variable
filter_notch_st 320()	R	-	F	Notch filter on the variable that can be used in a for loop
get_hw_info 326()	-	T	F	Gets information about the boards directly connected to the CPU
get_virt_ow 334()	R	T	F	Gets the state of virtual output word
in_range 350()	R	T	F	Returns the value in the specified range
inc 351()	R	T	F	Increments a variable by a specified amount

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
index_of 352()	R	T	F	Returns the index of the symbolic variable
lit 702	R	T	I	Associates a symbolic with a text string
loop_time 363()	R	T	F	Calculates the time duration between two successive calls
maximum 364()	R	T	F	Returns the higher of two values
master_filter 364()	R	T	F	Filters the speed of a master
minimum 412()	R	T	F	Returns the smaller of two values
normalize 488()	R	T	F	Returns normalized value between 0 and 1
onoff_filt 489()	R	T	F	Captures the filtered rising and falling edges of a condition
out_m 710()	R	T	I	Changes the state of the output channels
par_even 491()	R	T	F	Function that returns true if the number of bits in 1 of the byte passed to it is even
par_odd 492()	R	T	F	Function that returns true if the number of bits in 1 of the byte passed to it is odd
planar_comp_cp 493()	R	-	F	Applies inverse compensation in the management of the Planar Compensation Table

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
planar_comp_ip 494 ()	R	-	F	Applies direct compensation in the management of the Planar Compensation Table
pos_capture_cmd 495()	R	T	F	Function to hardware capture the dimension of an axis
pos_capture_sts 499()	R	T	F	Support function to pos_capture_cmd( )
ramp 530()	R	T	F	Ramps the input variable toward a target value
rise 536()	R	T	F	Captures the rising edge of a condition
rise_filt 537()	R	T	F	Captures the rising edge of a condition filtered
rtc_date 540()	R	T	F	Sets the date of the clock calender
rtc_time 540()	R	T	F	Sets the time of the clock calender
set_virt_iw 569()	R	T	F	Sets the state of virtual input word
set_version 748()	R	T	I	Writes the software version of the application
similar 575()	R	T	F	Compares two variables and determines if they are similar
statistics 595()	R	T	F	Performs statistics on the specified variable
store_nvr_nopf 595 ()	R	T	F	Saves the specified NVR in static memory
store_nvrr_nopf 596()	R	T	F	Saves the specified NVRR in static memory

© 2025 Robox SpA

Legend: R = allowed use in Rules - T = allowed use in Tasks - I = Instruction - F = Function (thus has a return value)				
Keyword	R	T	F/I	Description
time_micro 616()	R	T	F	Returns the time in microseconds since the motion control was turned on
virt_io 642()	-	T	F	Defines a virtual input and output word zone
write_dis 648()	R	T	F	Writes a string to the 4*20 display

Functions

Function abs()

This function returns the absolute value of the specified value.

Syntax	**i8|i16|i32|real abs (i8|i16|i32|real value)**
value	Function argument
Result	Returns the absolute value of *value*
Validity	Rule, Task
Note	

Examples

Example of use:

```
; the result of division 1230/(-5) is saved in the real register
rr(3)
rr(3) = 1230.0 / (-5)
; the absolute value of the contents in rr(3) is saved in the
register rr(4)
rr(4) = abs 133(rr(3))
; value of registers after instruction execution:
; rr(3) = -246
; rr(4) = 246
```

© 2025 Robox SpA

Function abs_mod()

This function returns the module of the input value, appropriately modified in the case of a negative input argument.

Syntax	**real abs_mod (real value, real step)**
value	Is the real unit whose module you want to calculate (can be a generic real expression)
step	Modulator value (can be a generic real expression, but must always give a positive value)
Result	Returns: Condition: *value* >= 0 — Value: mod(*value*, *step*) Condition: Otherwise — Value: mod(*value*, *step*) + *step*
Validity	Rule, Task
Notes	This formula allows the output to construct a "sawtooth" that extends uniformly even for negative values of the input unit

Examples

Example usage:

```
; the contents of register rr(1) is incremented by 1.
rr(1) = inc 351(rr(1))
; in rr(2) is saved the module of rr(1) with 234
rr(2) = abs_mod 134(rr(1), 234)
```

Function acos()

Performs the arcocosine of a real type value.

Syntax	**real acos (real value)**
value	Argument of the arcocosine (between -1 and 1).
Result	Returns the arcocosine of *value*. The result is returned in radians and is between 0 and K_PI 17
Validity	Rule, Task
Note	

⊞ Diagnostics

- Generate 9900 ill. Arg A:1 T:x ST:xxx if argument < -1 (where x = number of tasks and xxx number of steps)
- Generate 9900 ill. Arg A:2 T:x ST:xxx if argument > 1 (where x = number of tasks and xxx number of steps)

Examples

Example usage:

```
; in rr(10) is put the arcocosine of the value of rr(5) converted
from radians to degrees
rr(10) = todeg(acos(rr(5)))
; in rr(10) is put the arcocosine of 0.5
rr(10) = acos(0.5)
```

Function ah_cmd()

This function interacts with RTE alarm history file.

Syntax	**i32 ah_cmd (i32 cmdMsk)**
cmdMsk	Meaning bit mask:

Bit	Hexadecimal value	Description
0	0x00000001	Clear alarm history command
1 -:- 31	...	Reserved

Result	Returns:

Value	Description
0 [ACK_OK]	All commands have been acknowledged and executed
101 [NACK_ILL_ARGS]	Acknowledged commands have been executed, but there are unknown commands
133 [NACK_NOT_FOUND]	Alarm history not enabled

Validity	Rule, Task
Note	

Examples

© 2025 Robox SpA

Example of use:

```
; request for alarm history reset
_if(resetAH)
 ah_cmd 135(1)
```

Function ah_get_d()

This function provides information about the alarms stored in the alarm history with return of the information in a variable of type stru_ah_d 796 or stru_ah_d_ext 797.

Syntax	**i32 ah_get_d (i32 ahId, stru_ah_d| stru_ah_d_ext stru)**
ahId	Number of the ID of the alarm about which you want information (this type of information is obtained with ah_info 139())
stru	Name of the stru_ah_d 796 or stru_ah_d_ext 797 structure where RTE puts the response data
Result	Returns:

Value	Description
0 [ACK_OK]	OK result
107 [NACK_RES_BUSY]	Busy alarm history manager
127 [NACK_READ_ERROR]	Invalid data in retentive memory
133 [NACK_NOT_FOUND]	Alarm history not enabled
143 [NACK_DIFFERENT_DATA]	Requested data has been overwritten by a new alarm

Validity	Rule, Task
Note	

Examples

Example usage:

```
; variable declaration of type STRU_AH_D
stru_ah_d 796 sInfoAll

; ...
; request information about the id 123 with return of the information
in the structure sInfoAll
```

© 2025 Robox SpA

```
ah_get_d(123, sInfoAll)
```

Function ah_get_s()

This function provides information about the alarms stored in the alarm history with return of the information in a variable of type string.

Syntax	**i32 ah_get_s (i32 ahId, string str [, i32 lenMax=str])**
ahId	Number of the ID of the alarm about which you want information (this kind of information is obtained with ah_info())
str	Name of the variable of type string
lenMax	Maximum length of the string. (optional, defaults to the size of the string variable *str*)
Result	Returns:

Value	Description
0 [ACK_OK]	OK result
107 [NACK_RES_BUSY]	Busy alarm history manager
127 [NACK_READ_ERROR]	Invalid data in retentive memory
133 [NACK_NOT_FOUND]	Alarm history not enabled
143 [NACK_DIFFERENT_DATA]	Requested data has been overwritten by a new alarm

Validity	Rule, Task
Note	

If the result is positive, the string obtained has the following format : "n - d(dd) m(mm) y(yyyy) hh:ii:ss alarmText".

Parameter	Description
n	Id of the alarm (only the necessary digits)
dd	Day
mm	Month
yyyy	Year
hh	Time
ii	Minute

© 2025 Robox SpA

Parameter	Description
ss	Seconds
alarmText	Text of the alarm

Note: You can customize the date and time format via the predefined variable date_format.

Examples

Example usage:

```
bool get_alarm_info
string alarm_info
int result_ah_get
int id_alarm

lit NACK_RES_BUSY 107
lit NACK_READ_ERROR 127
lit NACK_NOT_FOUND 133
lit NACK_DIFFERENT_DATA 143

; ...
if(get_alarm_info)
 get_alarm_info = false
 result_ah_get = ah_get_s 137(id_alarm, alarm_info)

 if(not result_ah_get)
 ; everything okay
 inf_report 697(0x20, "Alarm info retrieved from AH with ah_get_s():")
 inf_report(0x20, alarm_info)
 else
 select(result_ah_get)

 case NACK_RES_BUSY
 inf_report(0x80, "ERROR: ah_get_s() AH handler busy")
 break

 case NACK_READ_ERROR
 inf_report(0x80, "ERROR: ah_get_s() error reading retentive memory")
 break

 case NACK_NOT_FOUND
 inf_report(0x80, "ERROR: ah_get_s() alarm history not enabled")
 break

 case NACK_DIFFERENT_DATA
 inf_report(0x80, "ERROR: ah_get_s() info requested have been
overwritten")
 break

 default
 str_format 601(rep_msg, "ERROR: ah_get_s() returned unkwon code %i",
result_ah_get)
 inf_report(0x80, rep_msg)
```

© 2025 Robox SpA

```
 endselect
 endif
endif
```

Function ah_info()

This function makes it possible to obtain information about the alarms stored in the alarm history with data return in the variable stru_ah_info 798.

Syntax	**i32 ah_info (stru_ah_info stru)**
stru	Name of the stru_ah_info 798 structure where RTE puts the response data
Result	Returns:

Value	Description
0 [ACK_OK]	OK result
133 [NACK_NOT_FOUND]	History not enabled

Validity	Rule, Task
Note	

Examples

Example of use:

```
bool get_ah_info
int result_ah_info
stru_ah_info 798 info_alarm_histrory
string rep_msg

; ...
if(get_ah_info)
    get_ah_info = false
    result_ah_info = ah_info 139(info_alarm_histrory)

    if(not result_ah_info)
        inf_report 697(0x20, "Alarm histrory info:")
        str_format 601(rep_msg, "AH old id: %i",
info_alarm_histrory.oldId)
        inf_report(0x20, rep_msg)
        str_format(rep_msg, "AH new id: %i",
info_alarm_histrory.newId)
        inf_report(0x20, rep_msg)
        str_format(rep_msg, "AH num al: %i", info_alarm_histrory.nal)
        inf_report(0x20, rep_msg)
```

© 2025 Robox SpA

```
        str_format(rep_msg, "AH size: %i",
info_alarm_histrory.ahSize)
        inf_report(0x20, rep_msg)
    else
        inf_report(0x80, "### Alarm history not enabled! ###")
    endif
endif
```

Function ah_log()

Function that allows messages to be added directly to the alarm history.

Syntax	**i32 ah_log (i32 code [, par1=0] [, par2=0] [, par3=0])**
code	Alarm number. Must be within the message validity range (10000-:-12999)
par1	First parameter passed to the message. (optional, default 0)
par2	Second parameter passed to the message. (optional, default 0)
par3	Third parameter passed to the message. (optional, default 0)
Result	Returns:

Value	Description
0 [ACK_OK]	OK result
101 [NACK_ILL_ARGS]	The alarm number is not in the expected range. In this case, alarm 9900 illegal Arg is generated
133 [NACK_NOT_FOUND]	History not enabled

Validity	Rule, Task
Note	The use of the three parameters and/or their relative types depend on how the message was configured

The structure of messages is equivalent to that of alarms (see user_alarm_file for configuration).

Examples

Example of use:

```
; in alarm definition file
10007 0  -1 0x23002000 "Info with a INT parameter [%d]" -4  ; in this
case, par1 is an INTEGER type

; in r3 software
int resoAHlog
int parameter

;...
    if(inp(1))
        parameter = 1
    else
        parameter = 2
    endif
resoAHMsg = ah_log(10007, parameter)
```

Function ah_log_us()

Function that allows messages with USV user string parameters to be added directly to the alarm history.

Syntax	**i32 ah_log_us (i32 nAl [, par1=0] [, par2=0] [, par3=0])**
nAl	Alarm number. Must be within the message validity range (10000-:-11999)
s1	User string to be passed as the first parameter. (optional, default 0)
s2	User string to be passed as the second parameter. (optional, default 0)
s3	User string to be passed as the third parameter. (optional, default 0)
Result	Returns:

Value	Description
0	Result OK
101	The alarm number is not in the expected range. In this case, alarm 9900 illegal Arg A:1 is generated
103	Alarm manager not active

Validity	Rule, Task

© 2025 Robox SpA

Syntax	**i32 ah_log_us (i32 nAl [, par1=0] [, par2=0] [, par3=0])**
System Requirements	From RTE 34.30.0
Note	

Examples

Example of use:

```
string s1
string s2
int result
...
; set an axis log
str_format (s1, "ax:%d Broken Brake", 3)
result = ah_log_us(11101, s1)
...
...
; set an axis log
str_format (s1, "ax:%d Broken Drive", 3)
result = ah_log_us(11101, s1)
...
...
; set log about Emergency push button pressed
str_format (s1, "Emergency Pressed")
...
str_format (s2, "near first door")
result = ah_log_us(11102, s1, s2)
...
str_format (s2, "end of line")
result = ah_log_us(11102, s1, s2)

; alarm definition in user file
11101 0 -1 0x23004000 "Log 1 <%s>" 17 0 0
11102 0 -1 0x23004000 "Log 2 <%s:%s>" 17 17 0
```

Function al2am()

This function provides information about the AM alarm mask, depending on the alarm number and axis number, if any.

Syntax	**i32 al2am (i32 nAl , i32 nAx, i32 whichInfo)**
nAl	Number of the alarm
nAx	Number of the axis (1-:-32) or 0 if the alarm is non-axis
whichInfo	Type of information about the desired alarm mask(AM):

© 2025 Robox SpA

Syntax	**i32 al2am (i32 nAl , i32 nAx, i32 whichInfo)**

Value *whichInfo*	Description
0	Status of the AM bit related to the alarm
1	Index of the AM bit related to the alarm
2	Index bit of the AM relative to the alarm
3	Index bits of the AM related to the alarm, in the form of a mask

Result

Returns:

Condition	Value	Description
Error	-1	Illegal alarm number
	-2	Illegal axis number
	-3	Type of information requested illegal
	-4	The specified alarm is not an axis alarm
	-5	The specified alarm is an axis alarm, or no AM is associated
	-6	The association to the AM bit is not defined
whichInfo = 0	0	Status of the AM bit related to the low alarm (i.e., alarm not present)
	1	State of the AM bit related to the high alarm (i.e., alarm present)

© 2025 Robox SpA

Syntax	**i32 al2am (i32 nAl , i32 nAx, i32 whichInfo)**

Condition	Value	Description
whichInfo = 1	>= 0	Index of the AM bit corresponding to the selected alarm
whichInfo = 2	>= 0	Bit of the index of the AM corresponding to the chosen alarm
whichInfo = 3	>= 0	Bits of the index of the AM corresponding to the chosen alarm, in the form of a mask

Validity	Rule, Task
Note	

Examples

Example usage:

```
; from task I call the function to get information about alarm 9900,
which has associated AM[49].27
result = al2am 142(9900, 0, whichInfo)

; possible cases
• whichInfo = 0 -> result = 1 if alarm 9900 is present, otherwise
  result = 0
• whichInfo = 1 -> result = 49
• whichInfo = 2 -> result = 27
• whichInfo = 3 -> result = 134217728 (0x8000000)
```

Example usage:

```
u32 SUM_MAC_DIAGNO, appo, mB
i32 nAX, iB
; ...
appo = 0
; ...
; I get the index of the bit of the ALARM TRANSDUCER (alarm 10)
iB = al2am 142(10, 1, 2)

for(nAX = 1 , nAX <= 6 , nAX)
 ; I check if the transducer alarm is present.
```

© 2025 Robox SpA

```
 if(AM(nAX).(iB))
 ; I set a bit indicating that at least one axis has transducer alarm
 appo.(10) = true
 break
 endif
endfor

; I get the index of the bit of the FOLLOWING ALARM (alarm 11)
iB = al2am(11, 1, 2)

for(nAX = 1 , nAX <= 6 , nAX)
 ; check if tracking alarm is present.
 if(AM(nAX).(iB))
 ; I set a bit indicating that at least one axis has tracking alarm
 appo.(11) = true
 break
 endif
endfor

; the information handling is the same as before, but now the mask is
used instead of the single bit
; I take the USER 1500 alarm bit in the form of a mask
mB = al2am(1500, 1, 3)

for(nAX = 1 , nAX <= 6 , nAX)
 ; check if alarm 1500 is present.
 if(AM(nAX) r_and mB)
 ; I set a bit indicating that at least one axis has alarm 1500
 appo.(12) = true
 break
 endif
endfor
```

Function alarm_reset()

This function allows you to delete, all or one in particular, alarms from the alarm stack.

Syntax	**bool alarm_reset ([i32 nAl=0] [, i32 nAx=0])**
nAl	Alarm number (1-:-9999). (optional, default 0)
nAx	Number of the axis in alarm (1-:-32). (optional, default 0)
Result	Returns:

Value	Description
TRUE	Alarm reset

© 2025 Robox SpA

Syntax	**bool alarm_reset ([i32 nAl=0] [, i32 nAx=0])**
	Value: FALSE — Description: Illegal parameters detected
Validity	Rule, Task
Note	Reset operation involves: • clearing the alarm in the stack (if present) and any compaction • resetting of the alarm bit in theAM • initialization of the filter time to access the alarm history(ah_filter_time()) • adjustment of the predefined variables al_pres(), al_stack_mask() and ax_in_alarm() (if necessary)

Diagnostics

- Generate 9900 ill. Arg A:1 T:x ST:xxx if alarm number is in field reserved for system (where x = task number and xxx step number)
- Generate 9900 ill. Arg A:2 T:x ST:xxx if any axis number is not in range 0-:-32 (where x = task number and xxx step number)

Examples

Usage examples:

```
; reset all alarms present
alarm_reset 145()
; reset alarm 123 for all axes
alarm_reset(123)
; reset alarm 199 for the axis 10 only
alarm_reset(199, 10)
; reset all alarms for axis 30
alarm_reset(0, 30)
```

Function alarm_reset_par()

This function allows an alarm to be reset by also specifying the value of one or more parameters.

Syntax	**bool alarm_reset_par ([i32 nAl=0] [, i32 nAx] [, u32 parMask] [, real par1] [, real par2=0] [, real par3=0])**
nAl	Alarm number (1-:-9999). (optional, default 0)

© 2025 Robox SpA

Syntax	**bool alarm_reset_par ([i32 nAl=0] [, i32 nAx] [, u32 parMask] [, real par1] [, real par2=0] [, real par3=0])**
nAx	Number of the axis in alarm (1-:-32). (optional, default 0)
parMask	Mask to indicate which parameters are affected:

Bit	Hexadecimal value	Description
0	0x00000001	None. Equivalent to the alarm_reset() 145 function. (default)
1	0x00000002	par1
2	0x00000004	par2
3	0x00000008	par1 and par2
4	0x00000010	par3
5	0x00000020	par1 and par3
6	0x00000040	par2 and par3
7	0x00000080	par1, par2 and par3

par1	Value of the first parameter of the alarm. (optional, default 0)
par2	Value of the second parameter of the alarm. (optional, default 0)
par3	Value of the third alarm parameter. (optional, default 0)
Result	Returns:

Value	Description
TRUE	Alarm reset
FALSE	Illegal parameters detected

Validity	Rule, Task
Notes	Reset operation involves: • clearing the alarm in the stack (if present) and any compaction • resetting of the alarm bit in theAM

© 2025 Robox SpA

Syntax	**bool alarm_reset_par ([i32 nAl=0] [, i32 nAx] [, u32 parMask] [, real par1] [, real par2=0] [, real par3=0])**
	• initialization of the filter time to access the alarm history(ah_filter_time()) • adjustment of the predefined variables al_pres(), al_stack_mask() and ax_in_alarm() (if necessary)

Diagnostics

Generate alarm 9900:

Code	Description
1	Illegal alarm number
443	Illegal alarm number
2	Illegal axis number
444	Illegal axis number
445	Illegal axle number
3	*nAl=0 nAx=0 parMask<>0*
446	*nAl=0 nAx=0 parMask<>0*
4	*nAl=0 nAx<>0 parMask<>0*
447	*nAl=0 nAx<>0 parMask<>0*
5	*nAl<>0 nAx=0(nAl* is axis alarm) *parMask<>0*
448	*nAl<>0 nAx=0(nAl* is axis alarm) *parMask<>0*

Examples

Example of use:

```
; sample code
int nAl, nAx, a
u32 parMask
real pAl1, pAl2, pAl3

; ...
a = alarm_reset_par 146(nAl, nAx, parMask, pAl1, pAl2, pAl3)
```

Function alarm_reset_par_us()

This function allows an alarm to be reset from the stack by specifying the value of one or more USV user string parameters.

All alarms in the stack that match the specified parameters will be removed.

If:

- the alarm number, axis number, and parMask are 0, a full stack reset is performed
- the alarm number and parMask are 0, all alarms in the specific axis are reset
- the axis number and parMask are 0 and the alarm number is an axis alarm, all alarms of the different axes are cleared
- the axis number and parMask are 0 and the alarm number is not an axis alarm, the specific alarm is reset
- the parMask is other than 0, the specific alarm is reset if the alarm number, axis number (including 0), and parameters match

<table>
<tr><td>Syntax</td><td>bool alarm_reset_par_us (i32 nAl, i32 nAx, u32 parMask [, string s1=0] [, string s2=0] [, string s3=0])</td></tr>
<tr><td>nAl</td><td>Alarm number (0 or 1-:-9999).</td></tr>
<tr><td>nAx</td><td>Number of the axis in alarm (0 or 1-:-32)</td></tr>
<tr><td>parMask</td><td>Mask to indicate which parameters are affected:
<table>
<tr><th>Bit</th><th>Hexadecimal value</th><th>Description</th></tr>
<tr><td>0</td><td>0x00000001</td><td>No USV parameter</td></tr>
<tr><td>1</td><td>0x00000002</td><td>s1</td></tr>
<tr><td>2</td><td>0x00000004</td><td>s2</td></tr>
<tr><td>3</td><td>0x00000008</td><td>s1 and s2</td></tr>
<tr><td>4</td><td>0x00000010</td><td>s3</td></tr>
<tr><td>5</td><td>0x00000020</td><td>s1 and s3</td></tr>
<tr><td>6</td><td>0x00000040</td><td>s2 and s3</td></tr>
<tr><td>7</td><td>0x00000080</td><td>s1, s2, and s3</td></tr>
</table></td></tr>
<tr><td>s1</td><td>User string to be used as the first parameter of the alarm. (optional, default 0)</td></tr>
<tr><td>s2</td><td>User string to be used as the second parameter of the alarm. (optional, default 0)</td></tr>
<tr><td>s3</td><td>User string to be used as the third parameter of the alarm. (optional, default 0)</td></tr>
<tr><td>Result</td><td>Returns:</td></tr>
</table>

© 2025 Robox SpA

Syntax	**bool alarm_reset_par_us (i32 nAl, i32 nAx, u32 parMask [, string s1=0] [, string s2=0] [, string s3=0])**
	<table><tr><th>Value</th><th>Description</th></tr><tr><td>TRUE</td><td>• alarm cleared • alarm disabled • axis not present</td></tr><tr><td>FALSE</td><td>• illegal alarm number • illegal or undefined axis number • alarm number or axis are 0 but parMask different from 0 • alarm has no USV parameters</td></tr></table>
Validity	Rule, Task
System Requirements	From RTE 34.30.0
Note	

Diagnostics

- Generate 9900 ill. Arg A:1 T:x ST:xxx if alarm number is not in the range of user alarms (where x = task number and xxx step number)
- Generate 9900 ill. Arg A:2 T:x ST:xxx if any axis number is invalid (where x = task number and xxx step number)
- Generate 9900 ill. Arg A:3 T:x ST:xxx if alarm and axis number are 0 but parMask different from 0 (where x = task number and xxx step number)
- Generate 9900 ill. Arg A:4 T:x ST:xxx if alarm number is 0 but axis number and parMask are different from 0 (where x = task number and xxx step number)
- Generate 9900 ill. Arg A:5 T:x ST:xxx if axis number is 0 but alarm number and parMask are different from 0 (where x = task number and xxx step number)
- Generate 9987 wrong reset_alarm funct. on alarm:xxx if the alarm does not have a defined USV (where xxx alarm number)

Examples

Example usage:

```
string s1
string s2
int result
...
; reset an axis mayor alarm
str_format 601 (s1, "Broken Brake")
```

```
result = alarm_reset_par_us(1101, 3, 1, s1)
...
...
; reset an axis mayor alarm
str_format (s1, "Broken Drive")
result = alarm_reset_par_us(1101, 3, 1, s1)
...
...
; reset all warnings about Emergency push button pressed
str_format (s1, "Emergency Pressed")
result = alarm_reset_par_us(4101, 0, 1, s1)
...
; reset specific warnings about Emergency push button pressed
...
str_format (s2, "near first door")
result = alarm_reset_par_us(4101, 0, 2, 0, s2)
...
str_format (s2, "end of line")
result = alarm_set_us(4101, 0, 2, 0, s2)

; alarm definition in user file
1101 -1 14 0x22001000 "Fault <%s>" 17 0 0
4101 53 24 0x22001000 "Warning <%s:%s>" 17 17 0
```

Function alarm_set()

This function allows an alarm to be entered into the alarm stack.

NOTE: To insert a message into the alarm history, use the ah_log() 140 function.

Syntax	**bool alarm_set (i32 nAl [, i32 nAx=0] [, real par1=0] [, real par2=0] [, real par3=0])**
nAl	Alarm number (1000-:-1999, 3000-:-3999, 4000-:-8999)
nAx	Axis number in alarm (1÷32). (optional, default 0)
par1	First parameter to be entered. (optional, default 0)
par2	Second parameter to be entered. (optional, default 0)
par3	Third parameter to be entered. (optional, default 0)
Result	Returns:

© 2025 Robox SpA

Syntax	**bool alarm_set (i32 nAl [, i32 nAx=0] [, real par1=0] [, real par2=0] [, real par3=0])**
	Value / Description TRUE: Operation performed correctly FALSE: Illegal parameters detected (alarm not entered)
Validity	Rule, Task
Note	However, the data types of optional parameters par1, par2, par3 depend on the type of alarm involved. From RTE 34.24.8 it is no longer possible to use this function to set warnings (9000÷9999)

NOTE: Refer to the user alarm file for alarm text association.

Diagnostics

- Generate 9900 ill. Arg A:1 T:x ST:xxx if alarm number is in field reserved for system (where x = task number and xxx step number)
- Generate 9900 ill. Arg A:2 T:x ST:xxx if any axis number is not in range 0-:-32 (where x = task number and xxx step number)
- Generate 9998 illegal axes number in alarm <nAll> if the alarm involves an axis number, but the axis was not specified on alarm generation

Examples

Example usage:

```
alarm_set(1100, 0, r(1), r(2), r(3))
alarm_set(5000, 1, ip(1), iv(1), ia(1))
alarm_set(4000, ps_1)

; the corresponding user_alarm_file.stp may be:
1100 37 0 0x02000100 "the value of R(1) is:[%d], of R(2) is:[%d]
while r(3):[%d]" -4 -4 -4
5000 53 0 0x02000100 "ip=[%f],iv=[%f], ia=[%f]" 8 8
1002 53 1 0x02000100 "powerset name= %s" 11
```

Function alarm_set_us()

This function allows an alarm with user string variable USV to be added to the alarm stack and alarm history.

The alarm will be added to the stack and history when:

© 2025 Robox SpA

- it is not already present in the stack
- has a prominent parameter flag imposed and the parameters are different

Syntax	**bool alarm_set_us (i32 nAl, i32 nAx, string s1 [, string s2=0] [, string s3=0])**
nAl	Alarm number (1000-:-1999, 3000-:-3999, 4000-:-8999)
nAx	Axis number (1-:-32). Put 0 if it is not an axis alarm
s1	First parameter to be entered as user string
s2	Second parameter to be entered as user string. (optional, default 0)
s3	Third parameter to be entered as user string. (optional, default 0)
Result	Returns: Value / Description TRUE: • alarm generated • alarm disabled • axis not present FALSE: • illegal alarm number • illegal axis number • alarm without USV parameters
Validity	Rule, Task
System Requirements	From RTE 34.30.0
Note	

Note: Refer to the user alarm file for alarm text association.

Diagnostics

- Generate 9900 ill. Arg A:1 T:x ST:xxx if alarm number is not in user alarm range (where x = task number and xxx step number)
- Generate 9900 ill. Arg A:2 T:x ST:xxx if any axis number is not in range 0-:-32 and sys_flag_2.17=0 (where x = task number and xxx step number)
- Generate 99 ill. Arg A:2 T:x ST:xxx if any axis number is not in range 0-:-32 and sys_flag_2.17=1 (where x = task number and xxx step number)
- Generate 9988 wrong set_alarm funct. on alarm:xxx if alarm does not have a defined USV (where xxx alarm number)

Examples

© 2025 Robox SpA

Example usage:

```
string s1
string s2
int result
...
; set an axis mayor alarm
str_format (s1, "Broken Brake")
result = alarm_set_us(1101, 3, s1)
...
...
; set an axis mayor alarm
str_format (s1, "Broken Drive")
result = alarm_set_us(1101, 3, s1)
...
...
; set warning about Emergency push button pressed
str_format (s1, "Emergency Pressed")
...
str_format (s2, "near first door")
result = alarm_set_us(4101, 0, s1, s2)
...
str_format (s2, "end of line")
result = alarm_set_us(4101, 0, s1, s2)

; alarm definition in user file
1101 -1 14 0x22001000 "Fault <%s>" 17 0 0
4101 53 24 0x22001000 "Warning <%s:%s>" 17 17 0
```

Function as_get_d()

This function provides information about the alarms stored in the alarm stack, with information returned in a variable of type stru_as_d or stru_as_d_ext.

Syntax	**bool as_get_d (i32 pos, stru_as_d| stru_as_d_ext stru)**
pos	Position in the stack (1-:-32)
stru	Name of the stru_as_d or stru_as_d_ext structure where RTE puts the response data
Result	Returns:

Value	Description
FALSE	Result not OK. It generates alarm 9900 Ill. arg. A:1 if the index of the position in the stack is out of limits

Syntax	**bool as_get_d (i32 pos, stru_as_d| stru_as_d_ext stru)**
	Value: TRUE Description: Result OK. Info about the alarm are in the structure
Validity	Rule, Task
Note	

Examples

Example usage:

```
; variable declaration of type STRU_AS_D
stru_as_d 799 sInfoAll

; ...
; request alarm at position 1 of the stack with return of information
in the structure named sInfoAll
as_get_d 154(1, sInfoAll)
```

Function asin()

This function performs the arcosine of a value of real type.

Syntax	**real asin (real val)**
val	Argument of the arcosine (between -1 and 1)
Result	Returns the arcosine of the argument *val*. The result is returned in radians and is between -K_PIMEZZI 17 and K_PIMEZZI 17
Validity	Rule, Task
Notes	

Diagnostics

- Generate 9900 ill. Arg A:1 T:x ST:xxx if argument < -1 (where x = number of tasks and xxx number of steps)
- Generate 9900 ill. Arg A:2 T:x ST:xxx if argument > 1 (where x = number of tasks and xxx number of steps)

Examples

Example usage:

© 2025 Robox SpA

```
; in rr(10) is put the arcosine of the value of rr(5) converted from
radians to degrees
rr(10) = todeg(asin(rr(5)))
; in rr(10) is put the arcosine of 0.5
rr(10) = asin(0.5)
```

Function atan()

Performs the arctangent of a value of real type.

Syntax	**real atan (real val)**
val	Argument of the arctangent
Result	Returns the arctangent of the argument *val.* The result is returned in radians and is between -K_PIMEZZI and K_PIMEZZI
Validity	Rule, Task
Note	

Examples

Example of use:

```
rr(10) = atan(rr(5))
; if rr(5) -> infinity rr(10) -> K_PIMEZZI

rr(10 ) = atan(1)
; rr(10) = 0.7853(K_PI/4)

rr(10 ) = atan(0)
; rr(10) = 0
```

Function atan2()

Performs the arctangent of the relationship between arg1 and arg2 (arg1/arg2).

Syntax	**real atan2 (real arg1, real arg2)**
arg1	First argument of the arctangent, i.e., sine value (or numerator)
arg2	Second argument of the arctangent, that is, value of the cosine (or denominator)
Result	In contrast to atan(), the function atan2 calculates a result between -K_PI and K_PI depending on the signs of its two arguments

Syntax	**real atan2 (real arg1, real arg2)**
Validity	Rule, Task
Note	

In the particular case where arg2 is equal to 0, the result will be:

- +K_PIMEZZI 17 if arg1 > 0
- -K_PIMEZZI 17 if arg1 < 0
- 0 if arg1 = 0

Examples

Example of use:

```
pippo = atan2 156(1,1)
; pippo = 0,7853 (K_PI 17/4)

pippo = atan2(0,1)
; pippo = 0

pippo = atan2(1,0)
; pippo = K_PIMEZZI 17
```

Esempio di utilizzo:

```
alfaRad = atan2 156(sinAlfa, cosAlfa)
betaDeg = todeg 621(atan2(120, 43))
```

Function axio_iol_r

This function allows reading a data area from a known master IOlink (AXL_SE_IOL4).

NOTE: In RDE, to date, there is no IOlink configurator.

Syntax	**i32 axio_iol_r (i32 rid, i32 slot, i32 buff[], i32 buffSize)**
rid	Robox ID
slot	Device location (1-:-63)
buff	Array of i32 where IOlink data will be read
buffSize	Size in number of bytes of buff
Result	Returns:

Value	Description
> 0	Operation performed successfully
-1	Illegal parameter: *red*

© 2025 Robox SpA

Syntax	**i32 axio_iol_r (i32 rid, i32 slot, i32 buff[], i32 buffSize)**
	Value / Description
	-2 / Illegal parameter: *slot*
	-3 / Illegal parameter: non-IOlink *slot*
	-4 / Illegal parameter: *buff* == NULL
	-5 / Read error
Validity	Task
Note	To understand the rid and slot parameters associated with modules, read the 'Configuration' section of axioline communication. The contents and actual length of the buffer depend only on the devices connected to the IOlink master. The meaning and format of individual data also depend only on the connected devices

Examples

Example usage:

```
; assumes an AXL_SE_IOL configured as.
; ch1 - digital input- it doesn't use any informtion in data area.
; ch2 - digital output- it doesn't use any informtion in data area
; ch3 - encoder (velocity - position)- it use 8 bytes in data area 4
bytes for velocity 4 bytes for position (both in big endian)
; ch4 - analog output - it use 2 bytes in data area (in big endian)
INT esitoiol
INT rid, slot ; rid and slot of device AXl_SE_IOL4
U8 bufiol[64] ; buffer used to store iolink information
INT encVel, encPos ;  information
...
...
esitoiol = AXIO_IOL_R (ridiol, slotiol, bufiol, sizeof(bufiol))
IF (esitoiol < 0)
; write here your diagnostic
ENDIF
; get now the information from the buffer
encVel = GET_I32(bufiol, 0); get velocity information from buffer in
Big Endian
encVel = DWBE2LE(encVel) ; convert information from Big Endian to
Little Endian (robox cpu format)
```

© 2025 Robox SpA

```
encPos = GET_I32(bufiol, 4); get position information from buffer in
Big Endian
encPos = DWBE2LE(encPos) ; convert information from Big Endian  to
Little Endian (robox cpu format)
```

Function axio_iol_w

This function allows a data area to be written to a known master IOlink (AXL_SE_IOL4).

NOTE: In RDE, to date, there is no IOlink configurator.

Syntax	**i32 axio_iol_r (i32 rid, i32 slot, i32 buff[], i32 buffSize)**
rid	Robox ID
slot	Device location (1-:-63)
buff	Array of i32 where IOlink data will be written
buffSize	Size in number of bytes of buff
Result	Returns:

Value	Description
> 0	Operation performed successfully
-1	Illegal parameter: *red*
-2	Illegal parameter: *slot*
-3	Illegal parameter: non-IOlink *slot*
-4	Illegal parameter: *buff* == NULL
-5	Write error

Validity	Task
Note	To understand the rid and slot parameters associated with modules, read the 'Configuration' section of axioline communication. The contents and actual length of the buffer depend only on the devices connected to the IOlink master. The meaning and format of individual data also depend only on the connected devices

Examples

© 2025 Robox SpA

Example usage:

```
; assumes an AXL_SE_IOL configured as.
; ch1 - digital input- it doesn't use any informtion in data area.
; ch2 - digital output- it doesn't use any informtion in data area
; ch3 - encoder (velocity - position)- it use 8 bytes in data area 4
bytes for velocity 4 bytes for position (both in big endian)
; ch4 - not used - it doesn't use any informtion in data area
INT esitoiol
INT rid, slot ; rid and slot of device AXl_SE_IOL4
U8 bufiolOut[64] ; buffer used to write to iolink
INT refOut ;  information to be written in the analog output
INT refoutD ; value reduced in the I16 value (+32767 -:- -32768)
I16 refoutBe
...
...
refOutD = IN_RANGE(refOut, -32758, 32767) ; reduce the value in 16
bits
refoutBe = refOutD ; store the informedin in the correct datatype
variable
refoutBe = WLE2BE (refoutBe) ;  convert from Little Endian to Big
Endian
SET_I16 (bufiolOut, 8, refOutBe) ; write to outpute data buffer
; write to IOLINK data output buffer
esitoiol = AXIO_IOL_W (ridiol, slotiol, bufiolOut, SIZEOF(bufiolOut))
IF (esitoiol < 0)
; write here your diagnostic
ENDIF
```

Function axio_pm_r()

This function allows you to read the data acquired from an AXL_F_PM_EF_1F device and save it in a stru_pm_pd 875 type ti structure or an i32 array.

The process information is read if the module is connected directly to the control or if it is connected to the EtherCAT head and then defined as device ECAT_PM.

NOTE: This instruction can be used only if the keyword ECAT_PM is included in rte.cfg.

Syntax	**i32 axio_pm_r (i32 rid, i32 slot, stru_pm_pd stru, i32 buffSize)** **i32 axio_pm_r (i32 rid, i32 slot, i32 buff[], i32 buffSize)**
rid	Robox ID
slot	Device location (1-:-63)
stru *buff*	Structure of type stru_pm_pd 875 or i32 array where data read from the device will be written

Syntax	**i32 axio_pm_r (i32 rid, i32 slot, stru_pm_pd stru, i32 buffSize)** **i32 axio_pm_r (i32 rid, i32 slot, i32 buff[], i32 buffSize)**
buffSize	Size in number of bytes of *buff*, must be a multiple of 4
Result	Returns: Value / Description > 0 / Operation performed successfully -1 / Illegal parameter: *rid* -2 / Illegal parameter: *slot* -3 / Illegal parameter: non-PM *slot* -4 / Illegal parameter: *buff* == NULL -5 / Illegal parameter: *buffSize* is not a multiple of 4 -6 / Error in read
Validity	Task
Note	To understand the rid and slot parameters associated with modules, read the 'Configuration' section of axioline communication

The data read by this function are as follows (with default settings):

Word	0, 1	2, 3	4, 5	6, 7	8, 9	10, 11	12, 13	14, 15	16, 17	18, 19	20, 21	22, 23	24, 25	26, 27	28, 29	30, 31
Phase	L1	L2	L3	L1	L2	L3	N	L1	L2	L3	L1	L2	L3	L1	L2	L3
Meaning	Voltage			Current				Active Power active energy			Reactive power reactive energy			Apparent power apparent energy		
Signal	L1_Voltage L2_Voltage L3_Voltage			L1_Current L2_Current L3_Current NeutralConductor_Current				L1_Real_PE L2_Real_PE L3_Real_PE			L1_Reactive_PE L2_Reactive_PE L3_Reactive_PE			L1_Apparent_PE L2_Apparent_PE L3_Apparent_PE		

© 2025 Robox SpA

Word	0, 1	2, 3	4, 5	6, 7	8, 9	10, 11	12, 13	14, 15	16, 17	18, 19	20, 21	22, 23	24, 25	26, 27	28, 29	30, 31
Unit	V			A				kW kWh			kVAr kVArh			kVA kVAh		
Resolution	1 mV			1 mA				1 W 1 Wh			1 VAr 1 VArh			1 VA 1 VAh		
Object settings (Default = bold)	Depending on object 0E11☐: 00☐: RMS value (RMS) 01☐: instantaneous value							Depending on object 0E12☐: 00☐: power 01☐: energy								
	Depending on object 0E13☐: 00☐:phase voltage 01☐: phase-to-phase voltage															

Examples

Example usage:

```
int pm_result
; rid and device slot AXl_SE_IOL4
int rid, slot
; buffer used to save data
stru_pm_pd 875 pm

; read measurements from module
pm_result = axio_pm_r 160(rid, slot, pm, sizeof(pm))
if(pm_result < 0)
 ; write your diagnostics here
endif

; check if there is more than 100V on each line.
if((pm.L1_voltage > 100000) && (pm.L2_voltage > 100000) &&
(pm.L2_voltage > 100000))
 ; ...
endif
```

Function axio_robj()

This function allows an object to be read from a specified axioline module.

Functioning

© 2025 Robox SpA

This function allows the object '*stru*.index + *stru*.subIndex' to be read on the axioline module identified by '*stru*.rid + *stru*.slot'.

The information received is stored in *destination*, which must be a variable large enough to hold the data that will be read. The size of the destination buffer must be specified in *stru*.dataLen.

Set *stru*.dataType = 0 to use the data type associated with the object (see table below of data type codes). In this case, the instruction will write in *stru*.dataType the data type code related to the object just read. Wanting to force a specific data type, set its value in *stru*.dataType. Normally, in the case of arrays you can read either the entire array (by specifying *stru*.subIndex = 0) or the individual element. When the reading is done, the instruction writes in *stru*.dataType the code for the data type used and in *stru*.nRic the number of bytes processed.

Reading can be done in pass-through mode or in blocking mode (usable only in tasks 1-:-8).

Pass-through mode means that the function must be continuously called and this returns the current state of the execution.

Blocking mode means that the function is invoked and returns only when it has completed (successfully or unsuccessfully) the read operation.

The function returns a result on the status of the execution.

Syntax	**i32 axio_robj (stru_axio_pdi stru, destination)**
stru	Structure name of type stru_axio_pdi 801
destination	Contains the received information. Must be an object compatible with the data being read (if reading a VISIBLE_STRING, *destination* will be of type string; if reading an OCTET_STRING, it will be a byte array or structure)
Result	Returns a result about the state of execution. See pass-through and/or blocking mode
Validity	Task
Notes	

Pass-through mode

- Set ***stru*.phase = 1** to initialize the read.
- The read command is sent to the remote device and a response is expected. The variable *stru*.phase is changed.
  The function returns 0.
- Waiting for response.
  The function continues to return 0.
- Receives response with **positive** result. The read value is written to *destination*. RTE changes *stru*.phase = 11.
  The function returns 1.
- Receive response with **negative** result. RTE changes *stru*.phase = 0.
  Function returns error code:

© 2025 Robox SpA

- -2, communication manager busy
- -5, command to a device not present or not axioline type ('rid')
- -6, no operation running. *stru*.phase not initialized
- -7, device information not yet available
- -8, internal error
- -10, illegal value
- -11, illegal value of *slot*
- -12, illegal value of *objCode*
- -13, illegal value of *subIndex*
- -14, illegal value of *dataType*
- -15, illegal value of *dataLen*
- -16, illegal value of phase(alarm 9900 arg 6)
- -17, error in read command (tunnel)
- -22, destination changed (phase management)
- 2, command for which a positive response has already been received, value available in *destination*
- >2, SDO error code, see table below

Blocking mode

- Set ***stru*.phase = -1** to perform the read.
- The read command is sent to the remote device and a response is expected. The variable *stru*.phase is changed.
- Waiting for response. If the response does not arrive within a maximum of 3 seconds, the function exits with an error (timeout).
- Receiving response with a **positive** result. The value read is written to *destination*. RTE changes *stru*.phase = 11.
  The function returns 1.
- Receive response with **negative** result. RTE changes *stru*.phase = 0.
  Function returns error code:
  - -2, communication manager busy
  - -4, blocking handler not allowed in current task*(stru*.phase = -1) (rule, ladder). Command not accepted (alarm 9900 arg 0)
  - -5, command to a device not present or not axioline type ('rid')
  - -6, no operation running. *stru*.phase not initialized
  - -7, device information not yet available
  - -8, internal error
  - -10, illegal value
  - -11, illegal value of *slot*
  - -12, illegal value of *objcode*

© 2025 Robox SpA

- -13, illegal value of *subIndex*
- -14, illegal value of *dataType*
- -15, illegal value of *dataLen*
- -16, illegal value of phase
- -22, changed destination (phase management)
- 2, command already answered positively, value available in *destination*
- >2, SDO error code, see table below

SDO Error Codes

SDO error code	Description
0x0503 0000	Toggle bit not toggled
0x0504 0000	SDO protocol timeout
0x0504 0001	Invalid or unknown client/server command identifier
0x0504 0002	Invalid block size (blocking mode only)
0x0504 0003	Invalid sequence number (blocking mode only)
0x0504 0004	CRC error (blocking mode only)
0x0504 0005	Memory overflow
0x0601 0000	Unsupported access to an object
0x0601 0001	Attempt to read a write-only object
0x0601 0002	Attempt to write a read-only object
0x0602 0000	Object does not exist in the object dictionary
0x0604 0041	Unable to map the object to the PDO
0x0604 0042	The number and length of objects to be mapped would exceed the length of the PDO
0x0604 0043	Parameter Incompatibility
0x0604 0047	Internal device incompatibility
0x0606 0000	Access failed due to hardware error
0x0607 0010	Data type does not match, length of service parameter does not match
0x0607 0012	Data type does not match, service duration parameter is too high

© 2025 Robox SpA

SDO error code	Description
0x0607 0013	Data type does not match, service duration parameter length is too low
0x0609 0011	Subindex does not exist
0x0609 0030	Field of parameter values exceeded (write access only)
0x0609 0031	Parameter value written too high
0x0609 0032	Parameter value written too low
0x0609 0036	Maximum value is less than minimum value
0x0800 0000	General error
0x0800 0020	Data cannot be transferred or stored in the application
0x0800 0021	Data cannot be transferred or stored in the application due to local control
0x0800 0022	Data cannot be transferred or stored in the application due to the current state of the device
0x0800 0023	Dynamic object dictionary generation fails or no object dictionary is present (e.g., object dictionary is generated from a file and generation fails due to a file error)

Data type codes (dataType)

Code	Data type
0	Automatic selection (default)
2	INTEGER8
3	INTEGER16
4	INTEGER32
5	UNSIGNED8
6	UNSIGNED16
7	UNSIGNED32
9	VISIBLE_STRING
10	OCTET_STRING
21	INTEGER64
27	UNSIGNED64

© 2025 Robox SpA

Code	Data type
-2	ARRAY INTEGER8
-3	ARRAY INTEGER16
-4	ARRAY INTEGER32
-5	ARRAY UNSIGNED8
-6	ARRAY UNSIGNED16
-7	ARRAY UNSIGNED32
-10	OCTET_STRING (used for special structures, e.g., parameter table CNT2/INC2)

Examples

Example of use:

```
; for AXIO_ROBJ / AXIO_WOBJ

int opAXIOobj                ; operation selection
                             ; 0  noone
                             ; 1 blocking read operation
                             ; -1 continuos blocking read operation
                             ; 2 pass-through read operation
                             ; -2  continuos pass-through read
operation
                             ; NOTE the read value will be store in
appropriate
                             ;      storage depending by pdi.dataType
see table below
                             ; 10 blocking write operation
                             ; -10 continuos blocking write operation
                             ; 11 pass-through write operation
                             ; -11  continuos pass-through write
operation

; desired parameter
int aRid                     ; RID of device where
int aSlot                    ; slot number of the module (from 1 up
63)
int aIndex                   ; PDI object index number
int aSubi                    ; PDI object subindex number
int aDataType = 0            ; dataType if 0  default data type are
assumed
int aDataLen = 0             ; in read if 0 di aBufIn sized is used in
write is not used
int aWhatWrite               ; specify what to write and where get the
information see table below
```

© 2025 Robox SpA

```
int aNElem                    ; number of array elements (if any)

; struct used to execute the instruction
stru_axio_pdi 801 pdi
u8  aBufIn[256]               ; general destination variable
real dtRAXIO                  ; used to compute the instruction
duration
real tActAxio                 ; instruction duration

; result of the operation
int resultAxioRObj
int resultAxioWObj
string a__srDatatype          ; managed dataType description

; source or destination variable
i8 a_I8                       ; used for INTEGER8
u8 a_U8                       ; used for UNSIGNED8
i16 a_I16                     ; used for INTEGER16
u16 a_U16                     ; used for UNSIGNED16
int a_I32                     ; used for INTEGER32
u32 a_U32                     ; used for UNSIGNED32
i16 a_aI16[256/2]             ; used for -INTEGER16 (ARRAY of
INTEGER16)
u16 a_aU16[256/2]             ; used for -UNSIGNED16 (ARRAY of
UNSIGNED16)
int a_aI32[256/4]             ; used for -INTEGER32 (ARRAY of
INTEGER32)
u32 a_aU32[256/4]             ; used for -UNSIGNED32 (ARRAY of
UNSIGNED32)
string a_VS                   ; used for VISIBLE_STRING
u8 a_Octet[256]               ; used for OCTET_STRING

struct_p 889 MVF
    float data
    u8    sts
    u8    unit
end_struct_p

MVF a_mvf                     ; used for -OCTET_STRING - object 0x82
subindex <> 0   MVF structure
MVF a_amvf[8]                 ; used for -OCTET_STRING - object 0x82
subindex = 0 (if possible)  MVF structure

struct_p CNT_INC_PARAMTABLE
    u16 basicConfig;
    u16 Resereved1;
    u16 outputConfig;
    u16 outputPulseTime;
    u32 startValue_convFact;
```

© 2025 Robox SpA

```
    u32 lowerLimit_moduloVal;
    u32 upperLimit_encIncVal;
    u32 reserved2_limitValOut1;
    u32 reserved3_limitValOut2;
    u32 reserved4_referenceVal;
    u32 reserved5_offsetZref;
    u32 reserved6_trasmFact;
    u32 reserved7_reserved8;
end_struct_p

CNT_INC_PARAMTABLE a_paraCntInc ; used for -OCTET_STRING - object
0x80 subindex <> 0   CNT_INC_PARAMTABLE structure

;<0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0>
;-----------------------------------------------------------------
function TestAXIOxOBJ()
;-----------------------------------------------------------------
;<0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0>

; function to test AXIO_ROBJE E AXIO_WOBJ instructions
select(opAXIOobj)
    case -100    ; error phase
        break

    case 0       ; nothing to do
        break

    case -1      ;opAxioObj = 1  continuous blocking Read on AXIO
device
    case 1       ;opAxioObj = 1  blocking Read on AXIO device
        pdi.rid      = aRid
        pdi.slot     = aSlot
        pdi.index    = aIndex
        pdi.subindex= aSubi
        pdi.dataType= aDataType
        pdi.dataLen = cond_val 265(aDataLen = 0, sizeof 576(aBufIn),
aDataLen)
        pdi.phase    = -1
        tActAxio     = time_micro 616()

        resultAxioRObj = axio_robj 162(pdi, abufin)
        dtRAXIO = time_micro() - tActAxio
        if(resultAxioRObj < 0)
            opAXIOobj = -100
            return
        endif

        ; prendiamo esito in funzione del datatype

        select(pdi.dataType)
            default    ; tipo non noto
```

```
            a__srDatatype = "DataType : non noto"
            break
        case 2     ; integer 8
            a__srDatatype = "DataType : integer 8"
            a_I8 = abufin[0]
            break
        case 5     ; unsigned 8
            a__srDatatype = "DataType : unsigned 8"
            a_U8 = abufin[0]
            break
        case 3     ; integer 16
            a__srDatatype = "DataType : integer 16"
            a_I16 = sw0 612 (abufin[0])
            break
        case -3    ; array integer 16
            a__srDatatype = "DataType : array integer 16"
            move_byte 704 (abufin, a_aI16, pdi.nric)
            break
        case 6     ; unsigned 16
            a__srDatatype = "DataType : unsigned 16"
            a_U16 = w0 644 (abufin[0])
            break
        case -6    ; array unsigned 16
            a__srDatatype = "DataType : array unsigned 16"
            move_byte(abufin, a_aU16, pdi.nric)
            break
        case 4     ; integer 32
            a__srDatatype = "DataType : integer 32"
            a_I32 = sdw0 557 (abufin[0])
            break
        case -4    ; array integer 32
            a__srDatatype = "DataType : array integer 32"
            move_byte(abufin, a_aI32, pdi.nric)
            break
        case 7     ; unsigned 32
            a__srDatatype = "DataType : unsigned 32"
            a_U32 = dw0 275 (abufin[0])
            break
        case -7    ; arrayunsigned 32
            a__srDatatype = "DataType : array unsigned 32"
            move_byte(abufin, a_aU32, pdi.nric)
            break
        case 9     ; visible string
            a__srDatatype = "DataType : visible string"
            move_byte(abufin, a_VS, pdi.nric)
            break
        case 10    ; Octet string
            a__srDatatype = "DataType : octet string"
            move_byte(abufin, a_Octet, pdi.nric)
            break
        case -10   ; -Octet string
```

© 2025 Robox SpA

```
                select(pdi.index)
                    default
                        a__srDatatype = "DataType : -Octet string -
unknown structure"
                        move_byte(abufin, a_Octet, pdi.nric)
                        break
                    case 0x80
                        a__srDatatype = "DataType : -Octet string -
CNT/INC param"
                        move_byte(abufin, a_paraCntInc, pdi.nric)
                        break
                    case 0x82
                        if(pdi.nRic > (int)sizeof(MVF))
                            a__srDatatype = "DataType : -Octet string
- MVF array"
                            move_byte(abufin, a_aMVF[0], pdi.nric)
                        else
                            a__srDatatype = "DataType : -Octet string
- MVF"
                            move_byte(abufin, a_MVF, pdi.nric)
                        endif
                end_select
                break
            case 27  ; U64
                a__srDatatype = "DataType : unsigned 64 as octet"
                move_byte(abufin, a_Octet, pdi.nric)
                break
        end_select

        _if(opAXIOobj> 0)
            opAXIOobj = 0;
        break
    case -2      ;opAxioObj = 1  continuous pass-through Read on AXIO
device
    case 2       ;opAxioObj = 1  pass-through Read on AXIO device
        pdi.rid      = aRid
        pdi.slot     = aSlot
        pdi.index    = aIndex
        pdi.subindex= aSubi
        pdi.dataType= aDataType
        pdi.dataLen = cond_val(aDataLen = 0, sizeof(aBufIn),
aDataLen)
        pdi.phase    = 1
        tActAxio     = time_micro();
    case -3
    case 3
        resultAxioRObj = axio_robj(pdi, abufin);
        if(resultAxioRObj = 0)
            opAXIOobj = cond_val(opAXIOobj < 0, -3, 3)
            break
        endif
        dtRAXIO = time_micro() - tActAxio;
```

© 2025 Robox SpA

```
        if(resultAxioRObj < 0)
            opAXIOobj = -100;
            return
        endif

        ; prendiamo esito in funzione del datatype

        select(pdi.dataType)
            default    ; tipo non noto
                a__srDatatype = "DataType : non noto"
                break
            case 2     ; integer 8
                a__srDatatype = "DataType : integer 8"
                a_I8 = abufin[0]
                break
            case 5     ; unsigned 8
                a__srDatatype = "DataType : unsigned 8"
                a_U8 = abufin[0]
                break
            case 3     ; integer 16
                a__srDatatype = "DataType : integer 16"
                a_I16 = sw0(abufin[0])
                break
            case -3    ; array integer 16
                a__srDatatype = "DataType : array integer 16"
                move_byte(abufin, a_aI16, pdi.nric)
                break
            case 6     ; unsigned 16
                a__srDatatype = "DataType : unsigned 16"
                a_U16 = w0(abufin[0])
                break
            case -6    ; array unsigned 16
                a__srDatatype = "DataType : array unsigned 16"
                move_byte(abufin, a_aU16, pdi.nric)
                break
            case 4     ; integer 32
                a__srDatatype = "DataType : integer 32"
                a_I32 = sdw0(abufin[0])
                break
            case -4    ; array integer 32
                a__srDatatype = "DataType : array integer 32"
                move_byte(abufin, a_aI32, pdi.nric)
                break
            case 7     ; unsigned 32
                a__srDatatype = "DataType : unsigned 32"
                a_U32 = dw0(abufin[0])
                break
            case -7    ; arrayunsigned 32
                a__srDatatype = "DataType : array unsigned 32"
                move_byte(abufin, a_aU32, pdi.nric)
                break
```

© 2025 Robox SpA

```
            case 9     ; visible string
                a__srDatatype = "DataType : visible string"
                move_byte(abufin, a_VS, pdi.nric)
                break
            case 10    ; Octet string
                a__srDatatype = "DataType : octet string"
                move_byte(abufin, a_Octet, pdi.nric)
                break
            case -10   ; -Octet string
                select(pdi.index)
                    default
                        a__srDatatype = "DataType : -Octet string -
unknown structure"
                        move_byte(abufin, a_Octet, pdi.nric)
                        break
                    case 0x80
                        a__srDatatype = "DataType : -Octet string -
CNT/INC param"
                        move_byte(abufin, a_paraCntInc, pdi.nric)
                        break
                    case 0x82
                        if(pdi.nRic > (int)sizeof(MVF))
                            a__srDatatype = "DataType : -Octet string
- MVF array"
                            move_byte(abufin, a_aMVF[0], pdi.nric)
                        else
                            a__srDatatype = "DataType : -Octet string
- MVF"
                            move_byte(abufin, a_MVF, pdi.nric)
                        endif
                end_select
                break
            case 27   ; U64
                a__srDatatype = "DataType : unsigned 64 as octet"
                move_byte(abufin, a_Octet, pdi.nric)
                break
        end_select

        if(opAXIOobj> 0)
            opAXIOobj = 0;
        else
            opAXIOobj = -2;
        endif
        break
    case -10         ;opAxioObj = -10  continuous blocking Write on
AXIO device
    case 10          ;opAxioObj = 10   blocking Write on AXIO device
        pdi.rid      = aRid
        pdi.slot     = aSlot
        pdi.index    = aIndex
        pdi.subindex= aSubi
        pdi.dataType= aDataType
```

© 2025 Robox SpA

```
        pdi.phase    = -1
        tActAxio     = time_micro();
        select(aWhatWrite)
            case 9     ; variable string
                a__srDatatype = "DataType : visible string"
                pdi.dataLen   = str_len(a_VS) +1
                resultAxioRObj   = axio_wobj(pdi, a_vs);
                break
            case 6     ; unsigned 16
                a__srDatatype = "DataType : unsigned 16"
                pdi.dataLen   = sizeof(a_U16)
                resultAxioRObj   = axio_wobj(pdi, a_u16);
                break
            case -6    ; array unsigned 16
                a__srDatatype = "DataType : array unsigned 16"
                pdi.dataLen   = sizeof(a_U16) * aNElem
                resultAxioRObj   = axio_wobj(pdi, a_au16);
                break
            case 3     ; integer 16
                a__srDatatype = "DataType : integer 16"
                pdi.dataLen   = sizeof(a_I16)
                resultAxioRObj   = axio_wobj(pdi, a_i16);
                break
            case -3    ; array integer 16
                a__srDatatype = "DataType : array integer 16"
                pdi.dataLen   = sizeof(a_I16) * aNElem
                resultAxioRObj   = axio_wobj(pdi, a_ai16);
                break
            case 5     ; unsigned 8
                a__srDatatype = "DataType : unsigned 8"
                pdi.dataLen   = sizeof(a_U8)
                resultAxioRObj   = axio_wobj(pdi, a_u8);
                break
            case 2     ; integer 8
                a__srDatatype = "DataType : integer 8"
                pdi.dataLen   = sizeof(a_I8)
                resultAxioRObj   = axio_wobj(pdi, a_i8);
                break
            case 10    ; Octet string
                a__srDatatype = "DataType : octet string"
                pdi.dataLen   = aNElem
                resultAxioRObj   = axio_wobj(pdi, a_octet);
                break

            case -10   ; -Octet string
                select(pdi.index)
                    default
                        a__srDatatype = "DataType : -Octet string -
unknown structure"
                        pdi.dataLen   = aNElem
                        resultAxioRObj   = axio_wobj(pdi, a_octet);
```

```
                        break
                    case 0x80
                        a__srDatatype = "DataType : -Octet string -
CNT/INC param"
                        pdi.dataLen  = sizeof(a_paraCntInc)
                        resultAxioRObj  = axio_wobj(pdi,
a_paracntinc);
                        break
                end_select
                break

        end_select

        dtRAXIO = time_micro() - tActAxio;
        if((resultAxioWObj < 0) or (resultAxioWObj > 2))
            opAXIOobj = -100;
            return
        endif
        _if (opAXIOobj> 0)
            opAXIOobj = 0;
        break   ; CASE 10 / -10


    case -11    ;opAxioObj = -11  continuous pass-through Write on
AXIO device
    case 11     ;opAxioObj = 11   pass-through Write on AXIO device
        pdi.rid     = aRid
        pdi.slot    = aSlot
        pdi.index   = aIndex
        pdi.subindex = aSubi
        pdi.dataType = aDataType
        pdi.phase   = 1
        tActAxio    = time_micro()
    case -12
    case 12
        select(aWhatWrite)
            case 9    ; variable string
                a__srDatatype = "DataType : visible string"
                pdi.dataLen  = str_len(a_VS) +1
                resultAxioRObj  = axio_wobj(pdi, a_vs);
                break
            case 6    ; unsigned 16
                a__srDatatype = "DataType : unsigned 16"
                pdi.dataLen  = sizeof(a_U16)
                resultAxioRObj  = axio_wobj(pdi, a_u16);
                break
            case -6   ; array unsigned 16
                a__srDatatype = "DataType : array unsigned 16"
                pdi.dataLen  = sizeof(a_U16) * aNElem
                resultAxioRObj  = axio_wobj(pdi, a_au16);
                break
            case 3    ; integer 16
```

© 2025 Robox SpA

```
                a__srDatatype = "DataType : integer 16"
                pdi.dataLen   = sizeof(a_I16)
                resultAxioRObj   = axio_wobj(pdi, a_i16);
                break
            case -3    ; array integer 16
                a__srDatatype = "DataType : array integer 16"
                pdi.dataLen   = sizeof(a_I16) * aNElem
                resultAxioRObj   = axio_wobj(pdi, a_ai16);
                break
            case 5     ; unsigned 8
                a__srDatatype = "DataType : unsigned 8"
                pdi.dataLen   = sizeof(a_U8)
                resultAxioRObj   = axio_wobj(pdi, a_u8);
                break
            case 2     ; integer 8
                a__srDatatype = "DataType : integer 8"
                pdi.dataLen   = sizeof(a_I8)
                resultAxioRObj   = axio_wobj(pdi, a_i8);
                break
            case 10    ; Octet string
                a__srDatatype = "DataType : octet string"
                pdi.dataLen   = aNElem
                resultAxioRObj   = axio_wobj(pdi, a_octet);
                break

            case -10   ; -Octet string
                select(pdi.index)
                    default
                        a__srDatatype = "DataType : -Octet string -
unknown structure"
                        pdi.dataLen   = aNElem
                        resultAxioRObj   = axio_wobj(pdi, a_octet);
                        break
                    case 0x80
                        a__srDatatype = "DataType : -Octet string -
CNT/INC param"
                        pdi.dataLen   = sizeof(a_paraCntInc)
                        resultAxioRObj   = axio_wobj(pdi,
a_paracntinc);
                        break
                end_select
                break

        end_select

        if(resultAxioWObj = 0)
            opAXIOobj = cond_val(opAXIOobj < 0, -12, 12)
            break
        endif
        dtRAXIO = time_micro() - tActAxio;
        if((resultAxioWObj < 0) or (resultAxioWObj > 2))
            opAXIOobj = -100;
```

© 2025 Robox SpA

```
            return
        endif
        if(opAXIOobj> 0)
            opAXIOobj = 0;
        else
            opAXIOobj =  -11
        endif
        break
    end_select
endfun
```

Function axio_wobj()

This function allows an object to be written to a specific axioline module.

Functioning

This function allows the object '*stru*.objCode + *stru*.subIndex' to be written to the axioline module identified by '*stru*.rid + *stru*.slot'.

The information present in *source* is written. The data in *source* will be taken according to the data type (see table below of data type codes). The data type, if not specified directly(*stru*.dataType <> 0), is inferred from the object it wants to write. The number of bytes to be written must be set in the field *stru*.dataLen. In case of single data types (integer8/16/32, unsigned8/16/32) it will be used to check if the source has sufficient size. In case of data types equal to OCTET_STRING it is taken as the number of bytes to be written. In case of VISIBLE_STRING the datatype is ignored and the size of the string is used directly. In the case of arrays, *stru*.datalen is used to calculate the number of elements in the array.

When written, the instruction imposes in *stru*.dataType the data type code used and in *stru*.nRic the number of bytes processed.

Normally, in the case of arrays it is possible to write either the entire array (by specifying *stru*.subIndex = 0) or the individual element.

Writing can be done in pass-through mode or in blocking mode (usable only in tasks 1-:-8). Blocking mode means that the function is invoked and returns only when it has completed (successfully or unsuccessfully) the write operation.

The function returns a result on the state of execution.

Syntax	**i32 axio_wobj (stru_axio_pdi stru, source)**
stru	Structure name of type stru_axio_pdi 801
source	Contains the information to be sent. *source* must be an object compatible with the data being written (if writing a VISIBLE STRING, *source* will be of type string; if writing an OCTET_STRING it will be a byte array or structure)
Result	Returns a result about the state of execution. See pass-through mode and/or blocking mode
Validity	Task

© 2025 Robox SpA

Syntax	**i32 axio_wobj (stru_axio_pdi stru, source)**
Note	

Pass-through mode

- Set ***stru*.phase = 1** in order to initialize the write operation.
- The command to write the *source* value is sent to the remote device and the response is awaited. The variable *stru*.phase is changed.
  The function returns 0.
- Waiting for response.
  The function continues to return 0.
- Receiving response with **positive** result. RTE changes *stru*.phase = 7.
  Function returns 1.
- Receiving response with **negative** result. RTE changes *stru*.phase = 0.
  Function returns error code:
  - -2, communication manager busy
  - -5, command to a device not present or not axioline type ('s.rid')
  - -6, no operation running. *stru*.phase not initialized
  - -7, device information not yet available
  - -8, internal error
  - -10, illegal value
  - -11, illegal value of *slot*
  - -12, illegal value of *objCode*
  - -13, illegal value of *subIndex*
  - -14, illegal value of *dataType*
  - -15, illegal value of *dataLen*
  - -16, illegal value of *phase* (alarm 9900 A:6)
  - -17, error in read command (tunnel)
  - -20, value written is the same
  - -23, read-only object
  - 2, command already positively answered, value available in *source*
  - >2, SDO error code, see below SDO error codes table

Blocking mode

- Set ***stru*.phase = -1** to execute the write.
- The command to write the *source* value is sent to the remote device and a response is awaited. The variable *stru*.phase is changed.
- Waiting for response. If the response does not arrive within a maximum of 3 seconds, the function exits with an error (timeout).

© 2025 Robox SpA

- Receiving response with **positive** result. RTE changes *stru*.phase = 7.
  The function returns 1.
- Receiving response with **negative** result. RTE changes *stru*.phase = 0.
  Function returns error code:
  - -2, communication manager busy
  - -4, blocking mode handler not allowed net current task(*stru*.phase = -1) (rule, ladder). Command not accepted(alarm 9900 A:0)
  - -5, command to a device not present or not axioline type(*stru*.rid)
  - -6, no operation running. *stru*.phase not initialized
  - -7, device information not yet available
  - -8, internal error
  - -10, illegal value
  - -11, illegal value of *slot*
  - -12, illegal value of *objCode*
  - -13, illegal value of *subIndex*
  - -14, illegal value of *dataType*
  - -15, illegal value of *dataLen*
  - -16, illegal value of *phase* (alarm 9900 A:6)
  - -17, error in read command (tunnel)
  - -20, value written is the same
  - -23, read-only object
  - 2, command already positively answered, value available in *source*
  - >2, SDO error code, see below SDO error codes table

SDO error codes

SDO error code	Description
0x0503 0000	Toggle bit not toggled
0x0504 0000	SDO protocol timeout
0x0504 0001	Invalid or unknown client/server command identifier
0x0504 0002	Invalid block size (blocking mode only)
0x0504 0003	Invalid sequence number (blocking mode only)
0x0504 0004	CRC error (blocking mode management only)
0x0504 0005	Memory overflow
0x0601 0000	Unsupported access to an object

© 2025 Robox SpA

SDO error code	Description
0x0601 0001	Attempt to read a write-only object
0x0601 0002	Attempt to write a read-only object
0x0602 0000	Object does not exist in the object dictionary
0x0604 0041	Unable to map the object to the PDO
0x0604 0042	The number and length of objects to map would exceed the length of the PDO
0x0604 0043	Incompatibility of parameters
0x0604 0047	Incompatibility internal to the device
0x0606 0000	Failed login due to hardware error
0x0607 0010	Data type does not match, length of service parameter does not match
0x0607 0012	Data type does not match, service parameter length is too high
0x0607 0013	Data type does not match, service duration parameter is too low
0x0609 0011	The subindex does not exist
0x0609 0030	Field of parameter values exceeded (write access only)
0x0609 0031	Parameter value written too high
0x0609 0032	Parameter value written too low
0x0609 0036	The maximum value is less than the minimum value
0x0800 0000	General error
0x0800 0020	Data cannot be transferred or stored in the application
0x0800 0021	Data cannot be transferred or stored in the application due to local control
0x0800 0022	Data cannot be transferred or stored in the application due to the current state of the device
0x0800 0023	Dynamic object dictionary generation fails or no object dictionary is present (for example, object dictionary is generated from a file and generation fails due to a file error)

Data type codes (dataType)

© 2025 Robox SpA

Code	DataType
0	Automatic selection (default)
2	INTEGER8
3	INTEGER16
4	INTEGER32
5	UNSIGNED8
6	UNSIGNED16
7	UNSIGNED32
9	VISIBLE_STRING
10	OCTET_STRING
21	INTEGER64
27	UNSIGNED64
-2	ARRAY INTEGER8
-3	ARRAY INTEGER16
-4	ARRAY INTEGER32
-5	ARRAY UNSIGNED8
-6	ARRAY UNSIGNED16
-7	ARRAY UNSIGNED32
-10	OCTET_STRING (used for special structures, e.g., parameter table CNT2/INC2)

Examples

Example of use:

```
; for AXIO_ROBJ / AXIO_WOBJ

int opAXIOobj                ; operation selection
                             ; 0  noone
                             ; 1 blocking read operation
                             ; -1 continuos blocking read operation
                             ; 2 pass-through read operation
                             ; -2  continuos pass-through read
operation
                             ; NOTE the read value will be store in
appropriate
                             ;      storage depending by pdi.dataType
see table below
```

© 2025 Robox SpA

```
                                ; 10 blocking write operation
                                ; -10 continuos blocking write operation
                                ; 11 pass-through write operation
                                ; -11  continuos pass-through write
operation

; desired parameter
int aRid                        ; RID of device where
int aSlot                       ; slot number of the module (from 1 up
63)
int aIndex                      ; PDI object index number
int aSubi                       ; PDI object subindex number
int aDataType = 0               ; dataType if 0  default data type are
assumed
int aDataLen = 0                ; in read if 0 di aBufIn sized is used in
write is not used
int aWhatWrite                  ; specify what to write and where get the
information see table below
int aNElem                      ; number of array elements (if any)

; struct used to execute the instruction
stru_axio_pdi 801 pdi
u8  aBufIn[256]                 ; general destination variable
real dtRAXIO                    ; used to compute the instruction
duration
real tActAxio                   ; instruction duration

; result of the operation
int resultAxioRObj
int resultAxioWObj
string a__srDatatype            ; managed dataType description

; source or destination variable
i8 a_I8                         ; used for INTEGER8
u8 a_U8                         ; used for UNSIGNED8
i16 a_I16                       ; used for INTEGER16
u16 a_U16                       ; used for UNSIGNED16
int a_I32                       ; used for INTEGER32
u32 a_U32                       ; used for UNSIGNED32
i16 a_aI16[256/2]               ; used for -INTEGER16 (ARRAY of
INTEGER16)
u16 a_aU16[256/2]               ; used for -UNSIGNED16 (ARRAY of
UNSIGNED16)
int a_aI32[256/4]               ; used for -INTEGER32 (ARRAY of
INTEGER32)
u32 a_aU32[256/4]               ; used for -UNSIGNED32 (ARRAY of
UNSIGNED32)
string a_VS                     ; used for VISIBLE_STRING
u8 a_Octet[256]                 ; used for OCTET_STRING
```

© 2025 Robox SpA

```
struct_p 889 MVF
    float data
    u8    sts
    u8    unit
end_struct_p

MVF a_mvf                    ; used for -OCTET_STRING - object 0x82
subindex <> 0   MVF structure
MVF a_amvf[8]                ; used for -OCTET_STRING - object 0x82
subindex = 0 (if possible)  MVF structure

struct_p CNT_INC_PARAMTABLE
    u16 basicConfig;
    u16 Resereved1;
    u16 outputConfig;
    u16 outputPulseTime;
    u32 startValue_convFact;
    u32 lowerLimit_moduloVal;
    u32 upperLimit_encIncVal;
    u32 reserved2_limitValOut1;
    u32 reserved3_limitValOut2;
    u32 reserved4_referenceVal;
    u32 reserved5_offsetZref;
    u32 reserved6_trasmFact;
    u32 reserved7_reserved8;
end_struct_p

CNT_INC_PARAMTABLE a_paraCntInc ; used for -OCTET_STRING - object
0x80 subindex <> 0   CNT_INC_PARAMTABLE structure

;<0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0>
;-----------------------------------------------------------------
function TestAXIOxOBJ()
;-----------------------------------------------------------------
;<0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0>

; function to test AXIO_ROBJE E AXIO_WOBJ instructions
select(opAXIOobj)
    case -100   ; error phase
        break

    case 0      ; nothing to do
        break

    case -1     ;opAxioObj = 1  continuous blocking Read on AXIO
device
    case 1      ;opAxioObj = 1  blocking Read on AXIO device
        pdi.rid     = aRid
        pdi.slot    = aSlot
```

© 2025 Robox SpA

```
        pdi.index    = aIndex
        pdi.subindex= aSubi
        pdi.dataType= aDataType
        pdi.dataLen = cond_val(aDataLen = 0, sizeof(aBufIn),
aDataLen)
        pdi.phase    = -1
        tActAxio     = time_micro()

        resultAxioRObj = axio_robj(pdi, abufin)
        dtRAXIO = time_micro() - tActAxio
        if(resultAxioRObj < 0)
            opAXIOobj = -100
            return
        endif

        ; prendiamo esito in funzione del datatype

        select(pdi.dataType)
            default    ; tipo non noto
                a__srDatatype = "DataType : non noto"
                break
            case 2     ; integer 8
                a__srDatatype = "DataType : integer 8"
                a_I8 = abufin[0]
                break
            case 5     ; unsigned 8
                a__srDatatype = "DataType : unsigned 8"
                a_U8 = abufin[0]
                break
            case 3     ; integer 16
                a__srDatatype = "DataType : integer 16"
                a_I16 = sw0(abufin[0])
                break
            case -3    ; array integer 16
                a__srDatatype = "DataType : array integer 16"
                move_byte(abufin, a_aI16, pdi.nric)
                break
            case 6     ; unsigned 16
                a__srDatatype = "DataType : unsigned 16"
                a_U16 = w0(abufin[0])
                break
            case -6    ; array unsigned 16
                a__srDatatype = "DataType : array unsigned 16"
                move_byte(abufin, a_aU16, pdi.nric)
                break
            case 4     ; integer 32
                a__srDatatype = "DataType : integer 32"
                a_I32 = sdw0(abufin[0])
                break
            case -4    ; array integer 32
                a__srDatatype = "DataType : array integer 32"
```

```
                move_byte(abufin, a_aI32, pdi.nric)
                break
            case 7     ; unsigned 32
                a__srDatatype = "DataType : unsigned 32"
                a_U32 = dw0(abufin[0])
                break
            case -7    ; arrayunsigned 32
                a__srDatatype = "DataType : array unsigned 32"
                move_byte(abufin, a_aU32, pdi.nric)
                break
            case 9     ; visible string
                a__srDatatype = "DataType : visible string"
                move_byte(abufin, a_VS, pdi.nric)
                break
            case 10    ; Octet string
                a__srDatatype = "DataType : octet string"
                move_byte(abufin, a_Octet, pdi.nric)
                break
            case -10   ; -Octet string
                select(pdi.index)
                    default
                        a__srDatatype = "DataType : -Octet string -
unknown structure"
                        move_byte(abufin, a_Octet, pdi.nric)
                        break
                    case 0x80
                        a__srDatatype = "DataType : -Octet string -
CNT/INC param"
                        move_byte(abufin, a_paraCntInc, pdi.nric)
                        break
                    case 0x82
                        if(pdi.nRic > (int)sizeof(MVF))
                            a__srDatatype = "DataType : -Octet string
- MVF array"
                            move_byte(abufin, a_aMVF[0], pdi.nric)
                        else
                            a__srDatatype = "DataType : -Octet string
- MVF"
                            move_byte(abufin, a_MVF, pdi.nric)
                        endif
                end_select
                break
            case 27   ; U64
                a__srDatatype = "DataType : unsigned 64 as octet"
                move_byte(abufin, a_Octet, pdi.nric)
                break
        end_select

        _if(opAXIOobj> 0)
            opAXIOobj = 0;
        break
```

© 2025 Robox SpA

```
    case -2      ;opAxioObj = 1  continuous pass-through Read on AXIO
device
    case 2       ;opAxioObj = 1  pass-through Read on AXIO device
        pdi.rid      = aRid
        pdi.slot     = aSlot
        pdi.index    = aIndex
        pdi.subindex= aSubi
        pdi.dataType= aDataType
        pdi.dataLen = cond_val(aDataLen = 0, sizeof(aBufIn),
aDataLen)
        pdi.phase    = 1
        tActAxio     = time_micro();
    case -3
    case 3
        resultAxioRObj = axio_robj(pdi, abufin);
        if(resultAxioRObj = 0)
            opAXIOobj = cond_val(opAXIOobj < 0, -3, 3)
            break
        endif
        dtRAXIO = time_micro() - tActAxio;
        if(resultAxioRObj < 0)
            opAXIOobj = -100;
            return
        endif

        ; prendiamo esito in funzione del datatype

        select(pdi.dataType)
            default    ; tipo non noto
                a__srDatatype = "DataType : non noto"
                break
            case 2     ; integer 8
                a__srDatatype = "DataType : integer 8"
                a_I8 = abufin[0]
                break
            case 5     ; unsigned 8
                a__srDatatype = "DataType : unsigned 8"
                a_U8 = abufin[0]
                break
            case 3     ; integer 16
                a__srDatatype = "DataType : integer 16"
                a_I16 = sw0(abufin[0])
                break
            case -3    ; array integer 16
                a__srDatatype = "DataType : array integer 16"
                move_byte(abufin, a_aI16, pdi.nric)
                break
            case 6     ; unsigned 16
                a__srDatatype = "DataType : unsigned 16"
                a_U16 = w0(abufin[0])
                break
```

© 2025 Robox SpA

```
            case -6    ; array unsigned 16
                a__srDatatype = "DataType : array unsigned 16"
                move_byte(abufin, a_aU16, pdi.nric)
                break
            case 4     ; integer 32
                a__srDatatype = "DataType : integer 32"
                a_I32 = sdw0(abufin[0])
                break
            case -4    ; array integer 32
                a__srDatatype = "DataType : array integer 32"
                move_byte(abufin, a_aI32, pdi.nric)
                break
            case 7     ; unsigned 32
                a__srDatatype = "DataType : unsigned 32"
                a_U32 = dw0(abufin[0])
                break
            case -7    ; arrayunsigned 32
                a__srDatatype = "DataType : array unsigned 32"
                move_byte(abufin, a_aU32, pdi.nric)
                break
            case 9     ; visible string
                a__srDatatype = "DataType : visible string"
                move_byte(abufin, a_VS, pdi.nric)
                break
            case 10    ; Octet string
                a__srDatatype = "DataType : octet string"
                move_byte(abufin, a_Octet, pdi.nric)
                break
            case -10   ; -Octet string
                select(pdi.index)
                    default
                        a__srDatatype = "DataType : -Octet string -
unknown structure"
                        move_byte(abufin, a_Octet, pdi.nric)
                        break
                    case 0x80
                        a__srDatatype = "DataType : -Octet string -
CNT/INC param"
                        move_byte(abufin, a_paraCntInc, pdi.nric)
                        break
                    case 0x82
                        if(pdi.nRic > (int)sizeof(MVF))
                            a__srDatatype = "DataType : -Octet string
- MVF array"
                            move_byte(abufin, a_aMVF[0], pdi.nric)
                        else
                            a__srDatatype = "DataType : -Octet string
- MVF"
                            move_byte(abufin, a_MVF, pdi.nric)
                        endif
                end_select
                break
```

© 2025 Robox SpA

```
            case 27   ; U64
                a__srDatatype = "DataType : unsigned 64 as octet"
                move_byte(abufin, a_Octet, pdi.nric)
                break
        end_select

        if(opAXIOobj> 0)
            opAXIOobj = 0;
        else
            opAXIOobj = -2;
        endif
        break
    case -10         ;opAxioObj = -10  continuous blocking Write on
AXIO device
    case 10          ;opAxioObj = 10   blocking Write on AXIO device
        pdi.rid      = aRid
        pdi.slot     = aSlot
        pdi.index    = aIndex
        pdi.subindex= aSubi
        pdi.dataType= aDataType
        pdi.phase    = -1
        tActAxio     = time_micro();
        select(aWhatWrite)
            case 9     ; variable string
                a__srDatatype = "DataType : visible string"
                pdi.dataLen   = str_len(a_VS) +1
                resultAxioRObj   = axio_wobj(pdi, a_vs);
                break
            case 6     ; unsigned 16
                a__srDatatype = "DataType : unsigned 16"
                pdi.dataLen   = sizeof(a_U16)
                resultAxioRObj   = axio_wobj(pdi, a_u16);
                break
            case -6    ; array unsigned 16
                a__srDatatype = "DataType : array unsigned 16"
                pdi.dataLen   = sizeof(a_U16) * aNElem
                resultAxioRObj   = axio_wobj(pdi, a_au16);
                break
            case 3     ; integer 16
                a__srDatatype = "DataType : integer 16"
                pdi.dataLen   = sizeof(a_I16)
                resultAxioRObj   = axio_wobj(pdi, a_i16);
                break
            case -3    ; array integer 16
                a__srDatatype = "DataType : array integer 16"
                pdi.dataLen   = sizeof(a_I16) * aNElem
                resultAxioRObj   = axio_wobj(pdi, a_ai16);
                break
            case 5     ; unsigned 8
                a__srDatatype = "DataType : unsigned 8"
                pdi.dataLen   = sizeof(a_U8)
```

```
                resultAxioRObj  = axio_wobj(pdi, a_u8);
                break
            case 2     ; integer 8
                a__srDatatype = "DataType : integer 8"
                pdi.dataLen  = sizeof(a_I8)
                resultAxioRObj  = axio_wobj(pdi, a_i8);
                break
            case 10    ; Octet string
                a__srDatatype = "DataType : octet string"
                pdi.dataLen  = aNElem
                resultAxioRObj  = axio_wobj(pdi, a_octet);
                break

            case -10   ; -Octet string
                select(pdi.index)
                    default
                        a__srDatatype = "DataType : -Octet string -
unknown structure"
                        pdi.dataLen  = aNElem
                        resultAxioRObj  = axio_wobj(pdi, a_octet);
                        break
                    case 0x80
                        a__srDatatype = "DataType : -Octet string -
CNT/INC param"
                        pdi.dataLen  = sizeof(a_paraCntInc)
                        resultAxioRObj  = axio_wobj(pdi,
a_paracntinc);
                        break
                end_select
                break

        end_select

        dtRAXIO = time_micro() - tActAxio;
        if((resultAxioWObj < 0) or (resultAxioWObj > 2))
            opAXIOobj = -100;
            return
        endif
        _if (opAXIOobj> 0)
            opAXIOobj = 0;
        break   ; CASE 10 / -10


    case -11    ;opAxioObj = -11  continuous pass-through Write on
AXIO device
    case 11     ;opAxioObj = 11   pass-through Write on AXIO device
        pdi.rid     = aRid
        pdi.slot    = aSlot
        pdi.index   = aIndex
        pdi.subindex = aSubi
        pdi.dataType = aDataType
        pdi.phase   = 1
```

© 2025 Robox SpA

```
        tActAxio     = time_micro()
    case -12
    case 12
        select(aWhatWrite)
            case 9     ; variable string
                a__srDatatype = "DataType : visible string"
                pdi.dataLen   = str_len(a_VS) +1
                resultAxioRObj   = axio_wobj(pdi, a_vs);
                break
            case 6     ; unsigned 16
                a__srDatatype = "DataType : unsigned 16"
                pdi.dataLen   = sizeof(a_U16)
                resultAxioRObj   = axio_wobj(pdi, a_u16);
                break
            case -6    ; array unsigned 16
                a__srDatatype = "DataType : array unsigned 16"
                pdi.dataLen   = sizeof(a_U16) * aNElem
                resultAxioRObj   = axio_wobj(pdi, a_au16);
                break
            case 3     ; integer 16
                a__srDatatype = "DataType : integer 16"
                pdi.dataLen   = sizeof(a_I16)
                resultAxioRObj   = axio_wobj(pdi, a_i16);
                break
            case -3    ; array integer 16
                a__srDatatype = "DataType : array integer 16"
                pdi.dataLen   = sizeof(a_I16) * aNElem
                resultAxioRObj   = axio_wobj(pdi, a_ai16);
                break
            case 5     ; unsigned 8
                a__srDatatype = "DataType : unsigned 8"
                pdi.dataLen   = sizeof(a_U8)
                resultAxioRObj   = axio_wobj(pdi, a_u8);
                break
            case 2     ; integer 8
                a__srDatatype = "DataType : integer 8"
                pdi.dataLen   = sizeof(a_I8)
                resultAxioRObj   = axio_wobj(pdi, a_i8);
                break
            case 10    ; Octet string
                a__srDatatype = "DataType : octet string"
                pdi.dataLen   = aNElem
                resultAxioRObj   = axio_wobj(pdi, a_octet);
                break

            case -10   ; -Octet string
                select(pdi.index)
                    default
                        a__srDatatype = "DataType : -Octet string -
unknown structure"
                        pdi.dataLen   = aNElem
```

© 2025 Robox SpA

```
                        resultAxioRObj   = axio_wobj(pdi, a_octet);
                        break
                    case 0x80
                        a__srDatatype = "DataType : -Octet string -
CNT/INC param"
                        pdi.dataLen   = sizeof(a_paraCntInc)
                        resultAxioRObj   = axio_wobj(pdi,
a_paracntinc);
                        break
                end_select
                break

        end_select

        if(resultAxioWObj = 0)
            opAXIOobj = cond_val(opAXIOobj < 0, -12, 12)
            break
        endif
        dtRAXIO = time_micro() - tActAxio;
        if((resultAxioWObj < 0) or (resultAxioWObj > 2))
            opAXIOobj = -100;
            return
        endif
        if(opAXIOobj> 0)
            opAXIOobj = 0;
        else
            opAXIOobj =  -11
        endif
        break
    end_select
endfun
```

Function b0()

This function is used to read or write (instruction) byte 0 of the argument.

Reading:

Syntax	**u8 b0 (i8|i16|i32|u8|u16|u32|real arg)**
arg	Variable whose contents you want to know. From RTE 33.7.1 it can be any expression
Result	Returns the contents of the first byte of the argument *arg*
Validity	Rule, Task
Note	From RTE 33.9.0 access to a non-existing byte returns error

Writing (instruction):

© 2025 Robox SpA

Syntax	**b0 (i8|i16|i32|u8|u16|u32|real arg) = u8 value**
arg	Variable in which the *value* value will be imposed
value	Value to be imposed
Validity	Rule, Task
Note	From RTE 33.9.0 access to a non-existing byte returns error

Examples

Example of use:

```
int pluto

; if pluto is worth 0x1234
r(1) = b0 191 (pluto) ; r(1)= 0x0034
r(1) = b1 192 (pluto) ; r(1)= 0x0012

;---------------------------------------------
; I define variable max as a 32 bit integer
int max
b0(max) = 0x11
b1(max) = 0x22
b2 193 (max) = 0x33
b3 194 (max) = 0x44
; eventually max will be = 0x44332211

;---------------------------------------------
; with RTE 33.9.0:
int pippo = 123
b4 194 (pippo) = 9
; error!
```

Function b1()

This function is used to read or write (instruction) byte 1 of the argument.

Reading:

Syntax	**u8 b1 (i16|i32|u16|u32|real arg)**
arg	Variable whose contents you want to know. From RTE 33.7.1 it can be any expression
Result	Returns the contents of the second byte of the argument *arg*

© 2025 Robox SpA

Syntax	**u8 b1 (i16|i32|u16|u32|real arg)**
Validity	Rule, Task
Note	From RTE 33.9.0 access to a non-existing byte returns error

Write (instruction):

Syntax	**b1 (i16|i32|u16|u32|real arg) = u8 value**
arg	Variable in which the *value* value will be imposed
value	Value to be imposed
Validity	Rule, Task
Note	From RTE 33.9.0 access to a non-existing byte returns error

Function b2()

This function is used to read or write (instruction) byte 2 of the argument.

Reading:

Syntax	**u8 b2 (i32|u32|real arg)**
arg	Variable whose contents you want to know. From RTE 33.7.1 it can be any expression
Result	Returns the contents of the third byte of the argument *arg*
Validity	Rule, Task
Note	From RTE 33.9.0 access to a non-existing byte returns error

Write (instruction):

Syntax	**b2 (i32|u32|real arg) = u8 value**
arg	Variable in which the *value* value will be imposed
value	Value to be imposed
Validity	Rule, Task
Note	From RTE 33.9.0 access to a non-existing byte returns error

© 2025 Robox SpA

Function b3()

This function is used to read or write (instruction) byte 3 of the argument.

Reading:

Syntax	**u8 b3 (i32|u32|real arg)**
arg	Variable whose contents you want to know. From RTE 33.7.1 it can be any expression
Result	Returns the contents of the fourth byte of the argument *arg*
Validity	Rule, Task
Note	From RTE 33.9.0 access to a non-existing byte returns error

Write (instruction):

Syntax	**b3 (i32|u32|real arg) = u8 value**
arg	Variable in which the *value* value will be imposed
value	Value to be imposed
Validity	Rule, Task
Note	From RTE 33.9.0 access to a non-existing byte returns error

Function b4()

This function is used to read or write (instruction) byte 4 of the argument.

Reading:

Syntax	**u8 b4 (real arg)**
arg	Variable whose contents you want to know. From RTE 33.7.1 it can be any expression
Result	Returns the contents of the fifth byte of the argument *arg*
Validity	Rule, Task
Note	From RTE 33.9.0 access to a non-existing byte returns error

Write (instruction):

Syntax	**b4 (real arg) = u8 value**
arg	Variable in which the *value* value will be imposed

© 2025 Robox SpA

Syntax	**b4 (real arg) = u8 value**
value	Value to be imposed
Validity	Rule, Task
Note	From RTE 33.9.0 access to a non-existing byte returns error

Function b5()

This function is used to read or write (instruction) byte 5 of the argument.

Reading:

Syntax	**u8 b5 (real arg)**
arg	Variable whose contents you want to know. From RTE 33.7.1 it can be any expression
Result	Returns the contents of the sixth byte of the argument *arg*
Validity	Rule, Task
Note	From RTE 33.9.0 access to a non-existing byte returns error

Write (instruction):

Syntax	**b5 (real arg) = u8 value**
arg	Variable in which the *value* value will be imposed
value	Value to be imposed
Validity	Rule, Task
Note	From RTE 33.9.0 access to a non-existing byte returns error

Function b6()

This function is used to read or write (instruction) byte 6 of the argument.

Reading:

Syntax	**u8 b6 (real arg)**
arg	Variable whose contents you want to know. From RTE 33.7.1 it can be any expression
Result	Returns the contents of the seventh byte of the argument *arg*
Validity	Rule, Task

© 2025 Robox SpA

Syntax	**u8 b6 (real arg)**
Note	From RTE 33.9.0 access to a non-existing byte returns error

Write (instruction):

Syntax	**b6 (real arg) = u8 value**
arg	Variable in which the *value* value will be imposed
value	Value to be imposed
Validity	Rule, Task
Note	From RTE 33.9.0 access to a non-existing byte returns error

Function b7()

This function is used to read or write (instruction) byte 7 of the argument.

Reading:

Syntax	**u8 b7 (real arg)**
arg	Variable whose contents you want to know. From RTE 33.7.1 it can be any expression
Result	Returns the contents of the eighth byte of the argument *arg*
Validity	Rule, Task
Note	From RTE 33.9.0 access to a non-existing byte returns error

Write (instruction):

Syntax	**b7 (real arg) = u8 value**
arg	Variable in which the *value* value will be imposed
value	Value to be imposed
Validity	Rule, Task
Note	From RTE 33.9.0 access to a non-existing byte returns error

© 2025 Robox SpA

Function bcc_get_double()

This function allows a real to be read from the *date* field of a bcc_msguser 793 structure at the specified offset.

Syntax	**real bcc_get_double (bcc_msguser stru, i32 offset)**
stru	Structure of type bcc_msguser 793
offset	Offset in number of bytes at which to read
Result	Returns the value read
Validity	Rule, Task
Note	

Examples

Example of use:

```
bcc_msguser bcc_str
real valueRead
int offset = 10

valueRead = bcc_get_double 197(bcc_str, offset)
```

Function bcc_get_float()

This function allows a float to be read from the *date* field of a bcc_msguser 793 structure at the specified offset.

Syntax	**float bcc_get_float (bcc_msguser stru, i32 offset)**
stru	Structure of type bcc_msguser 793
offset	Offset in number of bytes to be read
Result	Returns the value read
Validity	Rule, Task
Note	

Examples

Example of use:

```
bcc_msguser bcc_str
float valueRead
int offset = 10

valueRead = bcc_get_float 197(bcc_str, offset)
```

© 2025 Robox SpA

Function bcc_get_i16()

This function allows an i16 to be read from the *date* field of a bcc_msguser 793 structure at the specified offset.

Syntax	**i16 bcc_get_i16 (bcc_msguser stru, i32 offset)**
stru	Structure of type bcc_msguser 793
offset	Offset in number of bytes to be read
Result	Returns the value read
Validity	Rule, Task
Note	

Examples

Example of use:

```
bcc_msguser bcc_str
i16 valueRead
int offset = 10

valueRead = bcc_get_i16() 198(bcc_str, offset)
```

Function bcc_get_i32()

This function allows an i32 to be read from the *date* field of a bcc_msguser 793 structure at the specified offset.

Syntax	**i32 bcc_get_i32 (bcc_msguser stru, i32 offset)**
stru	Structure of type bcc_msguser 793
offset	Offset in number of bytes to be read
Result	Returns the value read
Validity	Rule, Task
Note	

Examples

Example of use:

```
bcc_msguser bcc_str
int valueRead
int offset = 10

valueRead = bcc_get_i32 198(bcc_str, offset)
```

© 2025 Robox SpA

Function bcc_get_i8()

This function allows an i8 to be read from the *date* field of a bcc_msguser 793 structure at the specified offset.

Syntax	**i8 bcc_get_i8 (bcc_msguser stru, i32 offset)**
stru	Structure of type bcc_msguser 793
offset	Offset in number of bytes to be read
Result	Returns the value read
Validity	Rule, Task
Note	

Examples

Example of use:

```
bcc_msguser bcc_str
i8 valueRead
int offset = 10

valueRead = bcc_get_i8 199(bcc_str, offset)
```

Function bcc_get_string()

This function allows a string to be read from the *date* field of a bcc_msguser 793 structure at the specified offset.

Syntax	**string bcc_get_string (bcc_msguser stru, i32 offset)**
stru	Structure of type bcc_msguser 793
offset	Offset in number of bytes to be read
Result	Returns the value read
Validity	Rule, Task
Note	

Examples

Example of use:

```
bcc_msguser bcc_str
string valueRead
int offset = 10

valueRead = bcc_get_string 199(bcc_str, offset)
```

© 2025 Robox SpA

Function bcc_get_tcp_clients_info()

Questa funzione permette di ottenere informazioni sui client connessi.

Syntax	**i32 bcc_get_tcp_clients_info (stru_bcc_tcp_client buff, u32 nElem [, u32 idx=0])**
buff	Structure of type stru_bcc_tccp_clients 803 or array of structures
nElem	Capacity of the array of structures
idx	Index of the offset in the client list. (optional, default 0)
Result	Returns:

Value	Description
>= 0	Number of client connected
-3	The size of the compiler structure does not match that of RTE
-4	*nElem* = 0
-5	Illegal value of *idx*

Validity	Rule, Task
Note	

Examples

Example of use:

```
lit MAX_TCP_CLIENTS 32

int num_of_client
stru_bcc_tcp_client 803 cl[MAX_TCP_CLIENTS]

; ...
num_of_client = bcc_get_tcp_clients_info 200(cl[0], MAX_TCP_CLIENTS)

if(num_of_client < 0)
    ; manage the error
    ; ...
endif
```

Function bcc_get_u16()

This function allows a u16 to be read from the *date* field of a bcc_msguser 793 structure at the specified offset.

Syntax	**u16 bcc_get_u16 (bcc_msguser stru, i32 offset)**
stru	Structure of type bcc_msguser 793
offset	Offset in number of bytes to be read
Result	Returns the value read
Validity	Rule, Task
Note	

Examples

Example of use:

```
bcc_msguser bcc_str
u16 valueRead
int offset = 10

valueRead = bcc_get_u16 201(bcc_str, offset)
```

Function bcc_get_u32()

This function allows a u32 to be read from the *date* field of a bcc_msguser 793 structure at the specified offset.

Syntax	**u32 bcc_get_u32 (bcc_msguser stru, i32 offset)**
stru	Structure of type bcc_msguser 793
offset	Offset in number of bytes to be read
Result	Returns the value read
Validity	Rule, Task
Note	

Examples

Example of use:

```
bcc_msguser bcc_str
u32 valueRead
int offset = 10

valueRead = bcc_get_u32 201(bcc_str, offset)
```

© 2025 Robox SpA

Function bcc_get_u8()

This function allows a u8 to be read from the *date* field of a bcc_msguser 793 structure at the specified offset.

Syntax	**u8 bcc_get_u8 (bcc_msguser stru, i32 offset)**
stru	Structure of type bcc_msguser 793
offset	Offset in number of bytes to be read
Result	Returns the value read
Validity	Rule, Task
Note	

Examples

Example of use:

```
bcc_msguser bcc_str
u8 valueRead
int offset = 10

valueRead = bcc_get_u8 202(bcc_str, offset)
```

Function bcc_h_setup()

This function performs the setting of data needed to respond to a BCC message.

Syntax	**bool bcc_h_setup (bcc_msguser struRx, bcc_msguser struTx, i32 len)**
struRx	Structure bcc_msguser 793 in which the message was received
struTx	Structure bcc_msguser 793 used for the data to be sent **NOTE**: The structure of the data to be sent can be the same as the data received.
len	Length of message data (max 255 bytes)
Result	Returns: Value — Description TRUE — Operation performed correctly FALSE — Illegal parameters detected
Validity	Rule, Task

Syntax	**bool bcc_h_setup (bcc_msguser struRx, bcc_msguser struTx, i32 len)**
Note	The operations performed are (logically) as follows: *struTx*.resSys1 = *struRx*.resSys1 *struTx*.resSys2 = *struRx*.resSys2 *struTx*.bccID = *struRx*.bccID *struTx*.DST = *struRx*.SR *struTx*.DCH = *struRx*.SCH *struTx*.SRC = *struRx*.DST *struTx*.SCH = *struRx*.DCH *struTx*.MSG = BCC_ACK (0x1) *struTx*.LEN = *len* *struTx*.PID = *struRx*.PID *struTx*.DLC = 0xFF

The response is sent to the source and the same physical device is used.

Examples

Example of use:

```
; let's see if there is a message on channel 190 to download
return = bcc_receive 204 (190, block) > 0
_if(return > 0)
 call 667 downloadData
goto loopT1
; end code task 1

;-----------------------------------------------
;-----------------------------------------------
; F U N C T I O N
;-----------------------------------------------
;-----------------------------------------------

function 688 downloadData
 ; copy the received data into the structure block.data
 ; starting with the integer regisro 2000 for 10 Bytes
 move_byte 704 (block.data, r(2000), 5*4)
 ; we make available to the master the register rr(1)
 ; which increments (watch-dog)
 inc 351 (rr(1))
 ; we copy the real register 1 into the date field of the
 ; block structure (RR = 8 bytes)
 move_byte(rr(1), block.data, 8)
 ; we respond to the master using the same header
 return = bcc_h_setup 202 (block, block, 8)
```

© 2025 Robox SpA

```
 return = bcc_send(190, block)
end_fun
```

Function bcc_receive()

This function allows incoming user messages to be read on predefined BCC3.0 channels.

Syntax	**i32 bcc_receive (i32 idCh, bcc_msguser stru)**
idCh	Channel index (190 ÷ 199)
stru	Locally declared bcc_msguser structure where the received message will be downloaded
Result	Returns:

Value	Description
> 0	Message received
0	No message received
-1, -2, -3	Illegal parameters
-4, -5	Illegal function (no device configured BCC3)

Validity	Rule, Task
Note	

Examples

Example usage:

```
; I define variable of type structure BCC_MSGUSER.
bcc_msguser block

returns = bcc_receive(190, block) > 0
_if(returns > 0)
 call downloadData
```

Function bcc_reply_ack()

This function allows a positive response (Ack) to be sent to a received message.

Syntax	**i32 bcc_reply_ack (bcc_msguser stru)**
stru	Bcc_msguser structure for handling BCC messages

© 2025 Robox SpA

Syntax	**i32 bcc_reply_ack (bcc_msguser stru)**
Result	Returns:

Value	Description
> 0	Message received
0	No message received
-1, -2, -3	Illegal parameters
-4, -5	Illegal function (no device configured BCC3)

Validity	Rule, Task
Note	

Function bcc_reply_msg()

This function allows you to send a reply to a received message.

Syntax	**i32 bcc_reply_msg (bcc_msguser struRx, bcc_msguser struTx, i32 code=0, i32 len=-1)**
struRx	Bcc_msguser 793 structure of bcc management of the message to be responded to
struTx	Bcc_msguser 793 structure of bcc management with the response information
code	If equal to 0, the message code (field *msg* of structure bcc_msguser) to be used in the response will be that of structure *struTx*, otherwise *code* will be used. (default 0)
len	If equal to -1, the length of the message (field *lendata* of the bcc_msguser structure) to be used in the response will be that of the *struTx* structure, otherwise *len* will be used. (default -1)
Result	Returns:

Value	Description
> 0	Message received
0	No message received
-1, -2, -3	Illegal parameters
-4, -5	Illegal function (no device configured BCC3)

© 2025 Robox SpA

Syntax	**i32 bcc_reply_msg (bcc_msguser struRx, bcc_msguser struTx, i32 code=0, i32 len=-1)**
Validity	Rule, Task
Note	

Function bcc_reply_nack()

This function allows you to send a negative (Nack) response to a received message.

Syntax	**i32 bcc_reply_nack (bcc_msguser stru, i32 nAck, i32 par=0)**
stru	Bcc_msguser 793 structure for handling BCC messages
nAck	Error code
par	Auxiliary parameter. (default 0)
Result	Returns:

Value	Description
> 0	Message received
0	No message received
-1, -2, -3	Illegal parameters
-4, -5	Illegal function (no device configured BCC3)

Validity	Rule, Task
Note	

Function bcc_send()

This function allows user messages to be sent on BCC3 default channels.

Syntax	**i32 bcc_send (i32 idCh, bcc_msguser stru)**
idCh	Channel index (190÷199)
stru	Locally declared bcc_msguser 793 structure from which the message to be sent will be read
Result	Returns:

© 2025 Robox SpA

Syntax	**i32 bcc_send (i32 idCh, bcc_msguser stru)**
	Value / Description > 0 / Message sent 0 / No message sent -1, -2, -3 / Illegal parameters -4, -5 / Illegal function (no device configured BCC3)
Validity	Rule, Task
Notes	For the message to be transmitted, the header data must have been entered correctly. This can be done with the ser_bcc_h_init 561() instruction

Examples

Example usage:

```
return = ser_bcc_h_init 561(block, 1, 190, 190, 10000, 5*4)
function 688 send
 ; copies from R(1000), 5*4 bytes i.e. 5 integer registers, into the
 ; date field of the block structure
 move_bytes 704(r(1000), block.data, 5*4)

 ; send the block structure on channel 190 (channels 190 to 199
available).
 ; return gives diagnostic information
  return = bcc_send 206(190, block)
end_fun
```

Function check_mv_follow2()

This function is used to check if mv_follow2 432() can be executed with the specified data. The function if it results in failure returns an error code; otherwise, if ok, it can also return the maximum synchronism duration time.

Syntax	**i32 check_mv_follow2 (stru_follow2 stru, real masterSpe, real syncTime)**
stru	Structure of type stru_follow2 819 containing movement-related parameters
masterSpe	Speed of the master at steady state
syncTime	Variable in which the maximum sync duration will be written
Result	Returns:

© 2025 Robox SpA

Syntax	**i32 check_mv_follow2 (stru_follow2 stru, real masterSpe, real syncTime)**

Valore	Descrizione
-5	Invalid axis number
-1	*masterSpe* <= 0
0	Operation performed successfully. You can run mv_follow2 432(), in *syncTime* you can read the maximum sync duration
1	Maximum acceleration of the slave <= 0
2	Start space (sync position / park position) or braking space are <= 0
3	*kSlaveMaster* <= 0
4	Rotary slave -> insufficient start master space + braking space
5	Rotary slave -> the master space available to the slave to reach the next sync is insufficient
	Linear slave -> the master space available to the slave to go back is insufficient
6	*masterCycle* <= 0
21	The slave exceeds the maximum speed while moving from the parking position to the sync position
22	The slave exceeds the maximum acceleration while moving from the parking position to the synchronism position
31	Linear slave -> the slave exceeds the maximum speed while braking after synchronism

© 2025 Robox SpA

Syntax	**i32 check_mv_follow2 (stru_follow2 stru, real masterSpe, real syncTime)**

Valore	Descrizione
32	Linear slave -> the slave exceeds the maximum acceleration during braking after synchronism
41	Linear slave -> the slave exceeds the maximum speed while returning to the initial position
42	Linear slave -> the slave exceeds the maximum acceleration while returning to the initial position
51	Rotary slave -> the slave exceeds the maximum speed while moving to the parking position after synchronism
52	Rotary slave -> slave exceeds maximum acceleration while moving to the parking position after synchronism
61	Rotary slave -> slave exceeds maximum speed while moving from the current synchronism position to the next one, slave stops to wait for the required master space
62	Rotary slave -> the slave exceeds the maximum acceleration while moving from the current synchronism position to the next one, the slave stops to wait for the necessary master space
71	Rotary slave -> the slave exceeds the maximum speed while moving from the current

© 2025 Robox SpA

Syntax	**i32 check_mv_follow2 (stru_follow2 stru, real masterSpe, real syncTime)**
	<table><tr><th>Valore</th><th>Descrizione</th></tr><tr><td></td><td>sync position to the next one</td></tr><tr><td>72</td><td>Rotary slave -> slave exceeds maximum acceleration as it moves from the current synchronism position to the next one</td></tr></table>
Validity	Rule, Task
Note	

Function check_mva_follow2()

This function is used to check if mva_follow2 432() can be executed with the specified data. The function if results in failure, returns an error code; otherwise, if ok, it can also return the maximum synchronism duration time.

Syntax	**i32 check_mva_follow2 (stru_follow2_s stru, real masterSpe, real syncTime)**
stru	Structure of type stru_follow2_s 827 containing motion-related parameters
masterSpe	Speed of the master at steady state
syncTime	Variable in which the maximum sync duration will be written
Result	Returns: <table><tr><th>Value</th><th>Description</th></tr><tr><td>-5</td><td>Invalid axis number</td></tr><tr><td>-1</td><td>*masterSpe* <= 0</td></tr><tr><td>0</td><td>Operation performed successfully. You can execute mva_follow2 432 (), in *syncTime* you can read the maximum sync duration</td></tr><tr><td>1</td><td>Maximum acceleration of the slave <= 0</td></tr><tr><td>2</td><td>Start space (sync position / park position) or braking space are <= 0</td></tr></table>

© 2025 Robox SpA

Syntax	**i32 check_mva_follow2 (stru_follow2_s stru, real masterSpe, real syncTime)**

Value	Description
3	*kSlaveMaster* <= 0
4	Rotary slave -> insufficient start master space + braking space
5	Rotary slave -> master space available to the slave to reach the next synchronism is insufficient
	Linear slave -> the master space available to the slave to go back is insufficient
6	*masterCycle* <= 0
21	The slave exceeds the maximum speed while moving from the parking position to the synchronous position
22	The slave exceeds the maximum acceleration while moving from the parking position to the synchronism position
31	Linear slave -> slave exceeds maximum speed while braking after synchronism
32	Linear slave -> the slave exceeds maximum acceleration while braking after synchronism
41	Linear slave -> the slave exceeds the maximum speed while returning to the initial position
42	Linear slave -> the slave exceeds the maximum acceleration while returning to the initial position

© 2025 Robox SpA

Syntax	**i32 check_mva_follow2 (stru_follow2_s stru, real masterSpe, real syncTime)**

Value	Description
51	Rotary slave -> the slave exceeds the maximum speed while moving to the parking position after synchronism
52	Rotary slave -> the slave exceeds maximum acceleration while moving to the parking position after synchronism
61	Rotary slave -> slave exceeds maximum speed while moving from the current synchronism position to the next, slave stops to wait for the required master space
62	Rotary slave -> slave exceeds maximum acceleration while moving from the current synchronism position to the next, slave stops to wait for the necessary master space
71	Rotary slave -> slave exceeds maximum speed as it moves from the current synchronization position to the next
72	Rotary slave -> slave exceeds maximum acceleration while moving from the current synchronism position to the next

Validity	Rule, Task
Note	

© 2025 Robox SpA

Function check_mva_follow2_m()

This function is used to check if mva_follow2_m 432() can be executed with the specified data. The function if results in failure, returns an error code; otherwise, if ok, it can also return the maximum synchronism duration time.

Syntax	**i32 check_mva_follow2 (stru_follow2_sm stru, real masterSpe, real syncTime)**
stru	Structure of type stru_follow2_sm 834 containing motion-related parameters
masterSpe	Speed of the master at steady state
syncTime	Variable in which the maximum sync duration will be written
Result	Returns:

Value	Description
-5	Invalid axis number
-1	*masterSpe* <= 0
0	Operation performed successfully. You can execute mva_follow2_m 432(), in *syncTime* you can read the maximum sync duration
1	Maximum acceleration of the slave <= 0
2	Start space (sync position / park position) or braking space are <= 0
3	*kSlaveMaster* <= 0
4	Rotary slave -> insufficient start master space + braking space
5	Rotary slave -> master space available to the slave to reach the next synchronism is insufficient
	Linear slave -> the master space available to the slave to go back is insufficient
6	*masterCycle* <= 0

© 2025 Robox SpA

Syntax	**i32 check_mva_follow2 (stru_follow2_sm stru, real masterSpe, real syncTime)**

Value	Description
21	The slave exceeds the maximum speed while moving from the parking position to the synchronous position
22	The slave exceeds the maximum acceleration while moving from the parking position to the synchronism position
31	Linear slave -> slave exceeds maximum speed while braking after synchronism
32	Linear slave -> the slave exceeds maximum acceleration while braking after synchronism
41	Linear slave -> the slave exceeds the maximum speed while returning to the initial position
42	Linear slave -> the slave exceeds the maximum acceleration while returning to the initial position
51	Rotary slave -> the slave exceeds the maximum speed while moving to the parking position after synchronism
52	Rotary slave -> the slave exceeds maximum acceleration while moving to the parking position after synchronism
61	Rotary slave -> slave exceeds maximum speed while moving from the current synchronism position to the next, slave stops to

© 2025 Robox SpA

Syntax	**i32 check_mva_follow2 (stru_follow2_sm stru, real masterSpe, real syncTime)**

Value	Description
	wait for the required master space
62	Rotary slave -> slave exceeds maximum acceleration while moving from the current synchronism position to the next, slave stops to wait for the necessary master space
71	Rotary slave -> slave exceeds maximum speed as it moves from the current synchronization position to the next
72	Rotary slave -> slave exceeds maximum acceleration while moving from the current synchronism position to the next

Validity	Rule, Task
Note	

Function co_asw_sdo()

WARNING: This function cannot be used in a task if the co_robj 249()/co_wobj()/coe_robj 255()/coe_wobj 259() functions are also present in the task.

This function allows you to:

- Test if the previous SDO command co_send_sdo 242(), used to write data, was executed -> co_asw_sdo(*nWs*, *handle*)
- Test if the previous SDO command co_send_sdo(), used to read data, was executed -> co_asw_sdo(*nWs*, *handle*)
- Test if the previous SDO command co_send_sdo(), used to read data, was executed and simultaneously to download data -> co_asw_sdo(*nWs*, *handle*, *type*, *data*)

Syntax	**i32 co_asw_sdo (i32 nWs, i32 handle)** **i32 co_asw_sdo (i32 nWs, i32 handle, i32 type, real data)**
nWs	Robox station ID (logical workstation 1-:-64).

© 2025 Robox SpA

Syntax	**i32 co_asw_sdo (i32 nWs, i32 handle)** **i32 co_asw_sdo (i32 nWs, i32 handle, i32 type, real data)**
	Note: If 0 is specified as ws then it is meant for all.
handle	Handle received from the previous execution of the co_send_sdo 242() function
type	Allows you to select how to interpret the data to be downloaded in *date*:

Value	Type
0	No object
1	Unsigned 8 bits
2	Unsigned 16 bits
3	Unsigned 24 bits
4	Unsigned 32 bits
5	Real 32 bits
-1	Signed 8 bits
-2	Signed 16 bits
-3	Signed 24 bits
-4	Signed 32 bits

data	Local variable where data is downloaded (from Fix format of specified size to real format) if *type* positive
Result	Returns:

Value	Description
-2	Message reference no longer valid
-1	Illegal data (alarm 9900 is generated)
0	Message waiting for response
1	Command executed and response obtained
> 1	Represents the SDO abort code according to the CANopen standard

© 2025 Robox SpA

Syntax	**i32 co_asw_sdo (i32 nWs, i32 handle)** **i32 co_asw_sdo (i32 nWs, i32 handle, i32 type, real data)**
Validity	Rule, Task
Note	From RTE 34.13.5, co_robj() and co_wobj() functions are available

Examples

Example of use:

```
; if there is a reading command
if(rise(command_read))
    command_read = false
    faseRead = SEND
endif

select(faseRead)
    case IDLE
        break

    case SEND
        handle = co_send_sdo(rNws, rObj, rSub)
        if(handle < 0)
            faseRead = ERROR
            break
        endif
        faseRead = GET_DATA
        break

    case GET_DATA
        result = co_asw_sdo(rNws, handle, rType, Data)
        ; waiting answer
        _if(result = 0)
            break
        ; test answer
        if(result = 1)
            faseRead = DONE
        else
            faseRead = ERROR
        endif
        break

    case DONE
        ; insert in the MREPORT diagnosys from SHELL
        str_format(srInfo, "SDO Read ws:%d, obj: %d, sub:%d,
returned value:%f", rnWs, rObj, rSub, Data)
        inf_report(0x1f,srInfo)
        faseRead = IDLE
        break
```

```
    case ERROR
        ; insert in the MREPORT diagnosys from SHELL
        str_format(srInfo, "Error SDO read value:%d", result)
        inf_report(0x1f, srInfo)
        faseRead = IDLE
        break
    end_select
```

Function co_controlword_om()

This function is used to impose the 'manufacturer specific', 'halt' and 'operation mode specific' bits of the CAN/CoE controlword (object 6040H).

Syntax	**i32 co_controlword_om (i32 nWs, i32 state, i32 bitMsk)**
nWs	Robox station ID (logical workstation 1-:-64). **Note**: If 0 is specified as *ws* then it is meant for all.
state	Value to be imposed on the specified bits
bitMsk	Mask of bits predisposed to assume the status
Result	Returns: Value: <> 0 — Successful operation Value: 0 — Illegal parameters detected
Validity	Rule, Task
Note	The mask of imposable bits with this instruction is 0x0000FD70, which corresponds to: Bit: 4, 5, 6 — Value Hex: 0x00000070 — Meaning: Operation mode specific Bit: 8 — Value Hex: 0x00000100 — Meaning: Halt Bit: 10 — Value Hex: 0x00000400 — Meaning: From RTE 34.16.6 Bit: 11, 12, 13, 14, 15 — Value Hex: 0x0000F800 — Meaning: Manufacturer specific Remember that in the rte.cfg configuration file, or from the graphical project configurator (axis properties), you can set the 'operation mode specific' bits via the axis_can keyword

© 2025 Robox SpA

co_controlword_om() for SOE devices

Since RTE 34.18.4, this function can also be used with SOE devices. The writable bits are 8, 9, 11 and the corresponding mask is 0x00000B00.

These bits correspond to:

Bits	Hex value	Meaning
-	0x00000000	Primary op mode
8	0x00000100	Secondary mode 1
9	0x00000200	Secondary mode 2
8, 9	0x00000300	Secondary mode 3
11	0x00000800	Secondary mode 4
8, 11	0x00000900	Secondary mode 5
9, 11	0x00000A00	Secondary mode 6
8, 9, 11	0x00000B00	Secondary mode 7

Controlword bits definition

Bits	15-14-13-12-11	10	9	8	7	6-5-4	3	2	1	0
Definition	Manufacturer specific	Reserved	Reserved	Halt	Fault reset	Operation mode specific	Enable operation	Quick stop	Enable voltage	Switch on
Bits imposable by co_controlword_om()	X	X		X		X				

Examples

Example of use:

```
; up transition of bit 0x0010
returned = co_controlword_om(1,0x0010,0x0010)
; down transition of bit 0x0010
returned = co_controlword_om(1,0x0000,0x0010)
```

Function co_controlword_state()

Function that imposes the status of the controlword.

Syntax	**i32 co_controlword_state (i32 nWs , i32 state)**
nWs	Robox Station ID (logical workstation 1-:-64). **Note**: If 0 is specified as *ws* then it is meant for all.
state	Possible states in accordance with DSP 402 state machine:

Value	Description
-2	DISABLE ENABLE (modal)
-1	POWER ON DISABLED (modal)
1	POWER ON ENABLED (modal)
2	ENABLE (modal but reset if you give quick stop)
3	DISABLE POWER (self-resetting)
4	QUICK STOP (self-resetting)
5	RESET FAULT (self-resetting)

Result	Returns:

Value	Description
<> 0	Successful operation
0	Illegal parameters encountered

Validity	Rule, Task
Note	The mask of imposable bits with this instruction is 0x00008F, which corresponds to:

Bit	Hex value	Meaning
0	0x00000001	Switch on
1	0x00000002	Enable voltage
2	0x00000004	Quick stop

© 2025 Robox SpA

Syntax	**i32 co_controlword_state (i32 nWs , i32 state)**

Bit	Hex value	Meaning
3	0x00000008	Enable operation
7	0x00000080	Fault reset

If the axis is managed using a PowerSet, the management of the DS402 state machine is done by RTE.

The use of this function is therefore necessary when:

- the management of a device is not done through a PowerSet
- to command a quick stop independently of RTE management
- to command a reset fault independently of RTE management

To know the current command present on the controlword use the co_statusword 246() function.

Controlword bits definition

Bits	15-14-13-12-11	10-9	8	7	6-5-4	3	2	1	0
Definition	Manufacturer specific	Reserved	Halt	Fault reset	Operation mode specific	Enable operation	Quick stop	Enable voltage	Switch on
Bits impossible by co_controlword_state()				X		X	X	X	X

Examples

Example of use:

```
; command, to all ws, to power on enable.
return = co_controlword_state 220(0, 1)
```

Function co_get_ws_diagno()

Function to read status information of the indicated workstation.

Syntax	**i32 co_get_ws_diagno (i32 nWs)**
nWs	Robox station ID (logical workstation 1-:-64).

Syntax	**i32 co_get_ws_diagno (i32 nWs)**
	Note: If 0 is specified as *ws* then it is meant for all.
Result	Returns the status code. The meaning of the bits that make up the status code is:

Bit	Hexadecimal value	Meaning
0	0x00000001	Rx PDO 1 missing.
1	0x00000002	Rx PDO 2 missing
2	0x00000004	Rx PDO 3 missing
3	0x00000008	Rx PDO 4 missing
4	0x00000010	Rx PDO 1 received before time
5	0x00000020	Rx PDO 2 received before time
6	0x00000040	Rx PDO 3 received before time
7	0x00000080	Rx PDO 4 received before time
8	0x00000100	Ws mandatory missing
9	0x00000200	Missing ws
10	0x00000400	Last RTR request failed
11	0x00000800	Ws presence imposed
12	0x00001000	Ws reconfiguration
13	0x00002000	Error during configuration
14	0x00004000	-
15	0x00008000	-

© 2025 Robox SpA

Syntax	i32 co_get_ws_diagno (i32 nWs)

Bit	Hexadecimal value	Meaning
16	0x00010000	Emergency message present
17	0x00020000	No communication (in operational mode)
18	0x00040000	Ws type: mandatory
19	0x00080000	Ws type: rechargeable at connection
20	0x00100000	Satisfactory configuration performed
21	0x00200000	Ws not connected to the first configuration
22	0x00400000	Problem during configuration of an SDO command
23	0x00800000	Problem during configuration of an NMT command
24	0x01000000	Configuration performed
25	0x02000000	SI in which data is read/write (only on physical CANbus devices)
26	0x04000000	The node is configured not present (ECAT_SLAVE_OFF)
27	0x08000000	A mode other than OPERATIONAL

© 2025 Robox SpA

Syntax	**i32 co_get_ws_diagno (i32 nWs)**
	<table><tr><th>Bit</th><th>Hexadecimal value</th><th>Meaning</th></tr><tr><td></td><td></td><td>was requested to the node</td></tr><tr><td>28</td><td>0x10000000</td><td>The EtherCAT master is not in OPERATIONAL</td></tr><tr><td>29</td><td>0x20000000</td><td>There is no link on EtherCAT master network port</td></tr><tr><td>30</td><td>0x40000000</td><td>Ws not started</td></tr><tr><td>31</td><td>0x80000000</td><td>Ws undefined</td></tr></table>
Validity	Rule, Task
Note	Returns the same information as the d_co_wsdiagno command from RDE shell

Examples

Example usage:

```
int canDiagno2, canDiagno3
canDiagno2 = co_get_ws_diagno(2)
canDiagno3 = co_get_ws_diagno(3)
```

Function co_par_download()

Function to upload to a device connected via CANopen DS301 the parameters contained in a file in the flash of the Robox control.

Functioning

The operations performed by this function are:

- Opening and analysis of the command file, where the analysis consists of:
  - Verifying existence of the "[FileInfo]" section.
  - Verifying existence of the "[OperationToDownloadParameters]" section.
  - Verification of the existence of the "[NodeInfo *ws*]" section and possible assumption of its commands "ParameterFileName=filename" and/or "Description=text"

  In the event that either of the first two points listed above are not met or an error is encountered, the function will fail and the error code will be returned.
- Identification of the file name from which to draw the parameters to be loaded according to the following criteria:

© 2025 Robox SpA

nameParFile	"ParameterFileName=filename"	File name
Present	Present	Same as the one specified in *nameParFile*
Present	Omitted	Same as the one specified in *nameParFile*
Omitted	Present	Same as the one specified in ParameterFileName
Omitted	Omitted	Same as the one specified in *nameCmdFile*

- Execution of the commands detected in the "[OperationToDownloadParameters]" section. Parameters are sent in the order in which the relevant objects were written to the file. The operational status of the command is returned in the predeclared variable sts_co_par_download, which of course can be used by the program after the execution of this step or viewed via the RDE tools even during execution. This variable can take the following values:

Value	Description
> 0	Line number of the file currently in use (command file or parameter file) in which the error causing the abnormal exit from the instruction was encountered
-1	Available executor wait
-2	Waiting for command execution
-3	Opening command file
-4	-
-5	-
-6	-
-7	Reading parameter files
-8	Parameter file command executions
-9	-

In the event that an error is encountered during parameter communication via SDO the error returned by the device is stored in the predeclared variable co_sdo_err (refer to device documentation for explanation of codes).

Note: The execution of this function may take a long time as it depends on the amount of data to be exchanged, the current traffic on the Can line, and the presence of other instructions being executed at the same level.

Syntax	**i32 co_par_download (string nameCmdFile, i32 nWs, i32 flagBrk [, string nameParFile="/FA/PAX_%d.DCF"])**
nameCmdFile	Full name of the command file

© 2025 Robox SpA

Syntax	**i32 co_par_download (string nameCmdFile, i32 nWs, i32 flagBrk [, string nameParFile="/FA/PAX_%d.DCF"])**
nWs	Workstation number on which you want to operate
flagBrk	Location to be used to abort the current execution. Any value different than 0 will cause the execution to abort. **WARNING**: The handling of aborting this function using this variable must be done in another task.
nameParFile	String containing "-F filename", where filename is the full name of the parameter file. (optional, default "/FA/PAX_%d.DCF")
Result	Meaning of the returned values:

Value	Description
< 0	NACK code not converted. (report to Robox)
0	Operation ok
1	Missing argument. (use mreport -a and/or the predeclared variable sts_co_par_download used for returns)
2	Illegal argument. (use mreport -a and/or sts_co_par_download)
3	One of the specified files does not exist. (use mreport -a)
4	One of the specified files is illegal. (use mreport -a)
5	Aborted operation. (use mreport -a)
6	Memory full (RAM or CF). (use mreport -a)
7	Unable to rename the file. File already exists. (use mreport -a)

© 2025 Robox SpA

Syntax	**i32 co_par_download (string nameCmdFile, i32 nWs, i32 flagBrk [, string nameParFile="/FA/PAX_%d.DCF"])**

Value	Description
8	Error in writing the file. (use mreport -a)
9	Error reading the file. (use mreport -a)
10	Unable to add more data to file in ram while handling. (use mreport -a)
11	Error on read/write object command on fieldbus. (use mreport -a and/or sts_co_par_download)
12	Error opening file. (use mreport -a)
13	Instruction/command handler busy
14	Operation ok but object or sub index not existing errors present. Occurs when disabling generation of return code 11 via sys_cfg. (use mreport -a)

Validity	Task
Note	

Function co_par_upload()

Function to update in the Robox control flash a parameter file, reading values directly from the drives via CANopen DS301. Starting from the objects described in the command file, a parameter file containing the definition and value of these objects is generated.

From RTE 34.31.0, setting sys_cfg_2.24 to 1 allows saving only those objects whose value is different from the *DefaultValue* specified in the template file. Along with these, objects with *AccessType=W* and those whose *DefaultValue* is not specified are also saved.

Functioning

The operations performed by this function are:

- Opening and analysis of the command file, where the analysis consists of:

© 2025 Robox SpA

- Verifying existence of the "[FileInfo]" section
- Verifying existence of the "[OperationToUploadParameters]" section
- Verification of the existence of the "[NodeInfo wsNumber]" section and possible assumption of its commands "ParameterFileName=filename" and/or "Description=text"

In the event that either of the first two points listed above is not met or an error is encountered, the function will fail and the error code will be returned.

- Identification of the file name from which to draw the parameters to be loaded according to the following criteria:

nameParFile	"ParameterFileName=filename"	File name
Present	Present	Same as the one specified in *nameParFile*
Present	Omitted	Same as that specified in *nameParFile*
Omitted	Present	Same as that specified in ParameterFileName
Omitted	Omitted	Same as PAXn.DCF

- Execution of the commands detected in the "[OperationToUploadParameters]" section. Parameters are read in the order in which the relevant objects were written to the command file. Only the objects in the [MandatoryObjects], [OptionalObjects] and [ManufacturerObjects] sections are taken. The operational status of the command is returned in the predeclared variable sts_co_par_upload, which of course can be used by the program after the execution of this step or viewed through the RDE tools even during execution. This variable can take the following values:

Value	Description
> 0	Line number of the file currently in use (command file or parameter file) in which the error causing the abnormal exit from the instruction was encountered
-1	Available executor wait
-2	Waiting for command execution
-3	Opening command file
-4	-
-5	-
-6	Analysis file upload commands
-7	Reading parameter files
-8	Executing parameter file commands
-9	Writing parameter file in progress

© 2025 Robox SpA

In the event that an error is encountered during parameter communication via SDO, the error returned by the device is stored in the predeclared variable co_sdo_err (refer to device documentation for explanation of codes).

Note: The execution of this function may take a long time as it depends on the amount of data to be exchanged, the current traffic on the Can line, and the presence of other instructions being executed at the same level.

Syntax	**i32 co_par_upload (string nameCmdFile, i32 nWs, i32 flagBrk [, string nameParFile="/FA/PAX_%d.DCF"] [, string comment=""])**
nameCmdFile	Full name of the command file
nWs	Number of workstations on which you want to operate
flagBrk	Location to be used to abort the current execution. Any value different than 0 will cause the execution to abort. **WARNING**: The handling of aborting this function using this variable must be done in another task.
nameParFile	String containing the name of the file to be included in the DCF file. (optional, default "/FA/PAX_%d.DCF"). The syntax for these parameters is "-F filename," where filename is the name of the parameters file complete with path
comment	String containing a comment to be inserted in the DCF file. (optional, default "") The syntax for these parameters is "-C comments" where comments are comments of up to 240 characters without LF (Line Feed)
Result	Meaning of the returned values:

Value	Description
< 0	NACK code not converted. (report to Robox)
0	Operation ok
1	Missing argument. (use mreport -a and/or the predeclared variable sts_co_par_download used for returns)
2	Illegal argument. (use mreport -a and/or

© 2025 Robox SpA

Syntax	**i32 co_par_upload (string nameCmdFile, i32 nWs, i32 flagBrk [, string nameParFile="/FA/PAX_%d.DCF"] [, string comment=""])**

Value	Description
	sts_co_par_download)
3	One of the specified files does not exist. (use mreport -a)
4	One of the specified files is illegal. (use mreport -a)
5	Aborted operation. (use mreport -a)
6	Memory full (RAM or CF). (use mreport -a)
7	Unable to rename the file. File already exists. (use mreport -a)
8	Error in writing the file. (use mreport -a)
9	Error reading the file. (use mreport -a)
10	Unable to add more data to file in ram while handling. (use mreport -a)
11	Error on read/write object command on fieldbus. (use mreport -a and/or sts_co_par_download)
12	Error opening file. (use mreport -a)
13	Instruction/command handler busy
14	Operation ok but non-existent object or sub index errors present. Occurs when disabling generation of return code 11 via sys_cfg. In case of upload, the

© 2025 Robox SpA

Syntax	**i32 co_par_upload (string nameCmdFile, i32 nWs, i32 flagBrk [, string nameParFile="/FA/PAX_%d.DCF"] [, string comment=""])**
	Value: Description: line with the value is inserted as a comment with the error report. (use mreport -a)
Validity	Task
Note	The flag sys_cfg_2.24, to enable the storage of only those objects whose value is different from *DefaultValue*, is retentive

Function co_rerr()

Function to read the last error message generated by the indicated ws.

Syntax	**i32 co_rerr (i32 nWs, stru_coerrmsg stru)**
nWs	Robox station ID (logical workstation 1-:-64)
stru	stru_coerrmsg 809 structure in which the information will be entered
Result	Returns:

Value	Description
0	No error message
1	Error message present
-1	Illegal parameters (number of illegal ws)

Validity	Rule, Task
Note	Returns the same information as the D_EMCY command from RDE shell

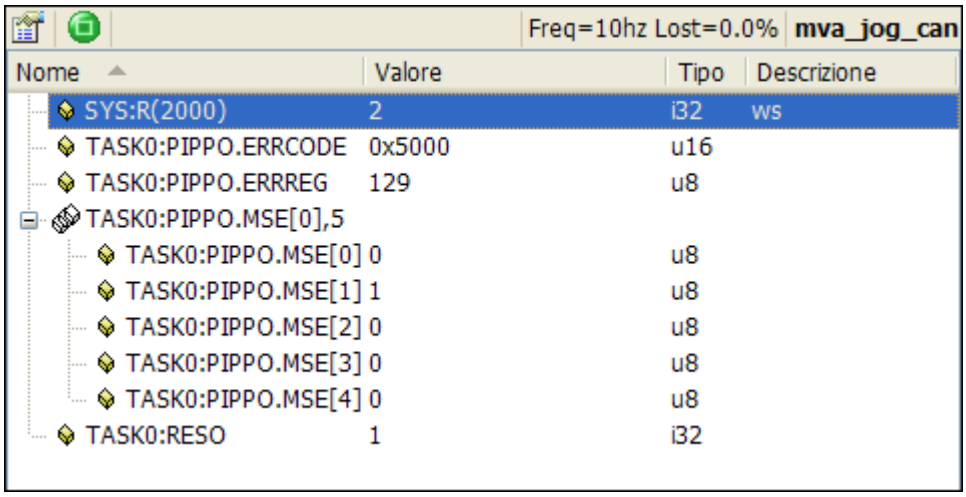
Examples

Example of use:

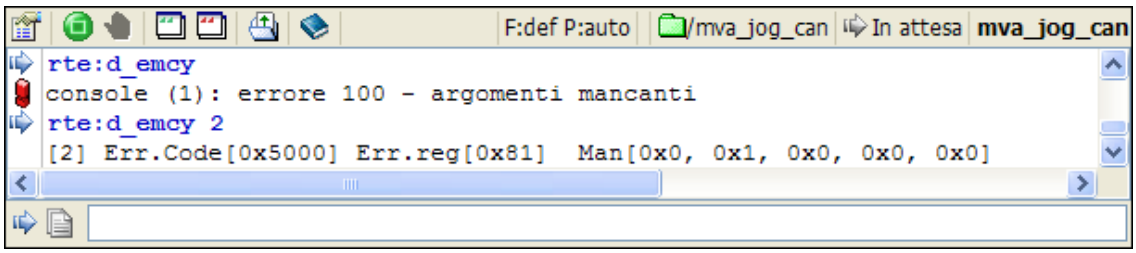
```
int result
stru_coerrmsg 809 struErr
```

```
result = co_rerr(r(2000), struErr)   ; in r(2000) logical ws is
specified
```

From RDE monitors it will be seen:



Sending the D_EMCY command from RDE shell gives the same information:



Function co_rnmt()

Function that returns the NMT operating mode of the indicated ws.

Syntax	**i32 co_rnmt (i32 nWs)**
nWs	ID stazione Robox (workstation logica 1-:-64)
Result	Returns the NMT state (first 8 bits):

Bit	Hex value	Meaning
-	0x00000000	BOOT
2	0x00000004	STOP
0, 2	0x00000005	OPERATIONAL
0, 1, 2, 3, 4, 5, 6	0x0000007f	PREOPERATIONAL

In addition, the following other information are mapped:

Syntax	**i32 co_rnmt (i32 nWs)**

Bit	Hex value	Meaning
8	0x00000100	=1 if station absent (always 0 if presence forced by cfg)
9	0x00000200	=1 if station really absent (even if it has forced presence)
10	0x00000400	=1 if the last request for a PDO via Remote Transmission Request (RTR) failed
11	0x00000800	=1 if station present because forced by configurator
12	0x00001000	=1 if station is being configured
13	0x00002000	=1 if the station aborted the configuration phase due to various causes
14	0x00004000	=1 if the station sent a shorter RPDO than the configured one
15	0x00008000	=1 if the station sent more than one synchronous PDO between two sync messages

Validity	Rule, Task
Note	Returns the same information as the D_NMT command from RDE shell

© 2025 Robox SpA

Function co_robj()

WARNING: This function cannot be used in a task if the co_send_sdo 242 ()/co_asw_sdo 215 () functions are also present in the task.

This function is used to perform reading of fieldbus objects. The read can be done in pass-through mode or in blocking mode (usable only in tasks 1-:-8).

Pass-through mode means that the function must be continuously called and simultaneously tested the current status of the task.

Blocking mode means that the function is invoked and returns only when it has completed the read operation.

The function returns a result on the state of execution.

Syntax	**i32 co_robj (stru_co_obj stru)**
stru	Structure name of type stru_co_obj 805
Result	Variable of type i32 containing the result. See pass-through mode and/or blocking mode management
Validity	Task, Rule (pass-through only)
Note	

Pass-through mode

- Set ***stru*.phase = 1** to initialize the read
- The read command is sent to the remote device and a response is expected. The variable *stru*.phase is changed.
  The function returns 0
- Waiting for response.
  The function continues to return 0
- Receives response with a **positive** result. The value read is written to *stru*.data. RTE changes *stru*.phase = 4.
  Function returns 1
- Receive response with **negative** result. RTE changes *stru*.phase = 0.
  Function returns error code:
  - -1, illegal parameters(alarm 9900)
  - -2, communication handler busy
  - -3, communication handler crash (restart)
  - -4, blocking mode not allowed in current task(*stru*.phase = 1) (rule, ladder). Command not accepted
  - -5, command to a device not present(*stru*.rid)
  - -6, no task running. *stru*.phase not initialized
  - 2, command already responded to positively, value available on *date*
  - >2, SDO error code, see table below

© 2025 Robox SpA

Blocking mode

- Set ***stru*.phase = -1** to initialize the read
- The read command is sent to the remote device and a response is expected. The variable *stru*.phase is changed
- Waiting for response. If the response does not arrive within a maximum of 3 seconds, the function exits with an error (timeout)
- Receiving response with a **positive** result. The value read is written to *stru*.data. RTE changes *stru*.phase = 4.
  The function returns 1
- Receive response with **negative** result. RTE changes *stru*.phase = 0.
  Function returns error code:
  - -1, illegal parameters (alarm 9900)
  - -2, communication handler busy
  - -3, communication handler crash (restart)
  - -4, blocking mode not allowed in current task*(stru*.phase = 1) (rule, ladder). Command not accepted
  - -5, command to a device not present*(stru*.rid)
  - -6, no task running. *stru*.phase not initialized
  - 2, command already responded to positively, value available in *stru*.data
  - >2, SDO error code, see table below

SDO SoE/CoE Error Codes

SDO error code	Description
0x0503 0000	Toggle bit not toggled
0x0504 0000	SDO protocol timeout
0x0504 0001	Invalid or unknown client/server command identifier
0x0504 0002	Invalid block size (blocking mode only)
0x0504 0003	Invalid sequence number (blocking mode only)
0x0504 0004	CRC error (blocking mode only)
0x0504 0005	Memory overflow
0x0601 0000	Unsupported access to an object
0x0601 0001	Attempt to read a write-only object
0x0601 0002	Attempt to write a read-only object
0x0602 0000	Object does not exist in the object dictionary

© 2025 Robox SpA

SDO error code	Description
0x0604 0041	Unable to map the object to the PDO
0x0604 0042	The number and length of objects to be mapped would exceed the length of the PDO
0x0604 0043	Incompatibility of parameters
0x0604 0047	Incompatibility internal to the device
0x0606 0000	Access failed due to hardware error
0x0607 0010	Data type does not match, length of service parameter does not match
0x0607 0012	Data type does not match, length of service parameter is too high
0x0607 0013	Data type does not match, service duration parameter is too low
0x0609 0011	Subindex does not exist
0x0609 0030	Field of parameter values exceeded (write access only)
0x0609 0031	Parameter value written too high
0x0609 0032	Parameter value written too low
0x0609 0036	Maximum value is less than minimum value
0x0800 0000	General error
0x0800 0020	Data cannot be transferred or stored in the application
0x0800 0021	Data cannot be transferred or stored in the application due to local control
0x0800 0022	Data cannot be transferred or stored in the application due to the current state of the device
0x0800 0023	Dynamic object dictionary generation fails or no object dictionary is present (e.g., object dictionary is generated from a file and generation fails due to a file error)

Examples

Example of use:

```
; SAMPLE CODE
; Global Storage definition
;--------------------------
```

© 2025 Robox SpA

```
real CpuDrive_temp[32+1]
real Heater_temp[32+1]
real ControlBoard_temp[32+1]
real updateTime[32+1]
; storage used only by ReadImdTempPT
int whatTemp[32+1]
stru_co_obj 805 sobj[32+1]
lit PER_UPD_TEMP 1.0
..
..
..
; Before the main loop
call InitReadImdTemp()
...
...
; inside the main loop
if(IWantPassThroughReadTempRoutine) ; test a flag to check if i have
to use pass-through
                                    ; or blocking routines
    result = ReadImdTempPT(k) ; k is the RID of the IMD that I want
to question
else
    result = ReadImdTempBL(k) ; k is the RID of the IMD that I want
to question
endif
...
...
...

;<0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0>
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
FUNCTION InitReadImdTemp()
    real tAct
    int k
    tAct = tfb
    ; initialize the tome to start reading after 5 seconds
    ; in sequence way
    for(k=1, k <=32 , INC(k))
        updateTime[k] = tAct + PER_UPD_TEMP + ((k-1)
*PER_UPD_TEMP/32.0)
        whatTemp[k] = 0
    end_for
ENDFUN

;<0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0>
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
FUNCTION int ReadImdTempPT(int rid)
    ; routine 'PASS_THROUGH' version
```

© 2025 Robox SpA

```
    ; function to update the IMD temperature value each PER_UPD_TEMP
seconds.
    ; the information will be set in global storage.
    ; the function want the RID of the IMD
    ; give back:
    ; <0 error
    ; >2 error (SDO diagnostic)
    ; 0 no operation
    ; 1 operation in progress
    ; 2 all temperature updated
    int ero
    ; check RID value
    if(not range(rid, 1, 32))
        return -1 ; return error if illegal value
    endif
    ; check if time to update
    if(updateTime[rid] <= tfb)
        inc(updateTime[rid], PER_UPD_TEMP)
        if(whatTemp[rid] = 0)
            whatTemp[rid] = 1
        endif
    endif
    ; check what we have to do
    select(whatTemp[rid])
        case 0
            return 0 ; nothing to do
        case 1
            sObj[rid].rid = rid
            sObj[rid].objCode = 0x2101 ; Cpu drive Temp
            sObj[rid].subIndex = 2
            sObj[rid].type = 5 ; float data
            sObj[rid].phase = 1 ; set PASS_THROUGH modality
            whatTemp[rid] = 2 ; next step
            ; break intentionally omitted
        case 2
            ero = co_robj(sobj[rid])
            select(ero)
                default
                    whatTemp[rid] = 0 ; go back
                    return ero ; return error code
                case 0
                    return 1 ; we have to wait
                case -2 ; busy, restart again next loop
                    sObj[rid].phase = 1 ; set PASS_THROUGH modality
for the next call
                    return 1
                case 1 ; we got the answer and we save it
                    CpuDrive_temp[rid] = sObj[rid].data
                    break
```

```
            end_select
            ; prepare the read next temperature
            sObj[rid].objCode = 0x2101 ; Heater Temp
            sObj[rid].subIndex = 1
            sObj[rid].type = 5 ; float data
            sObj[rid].phase = 1 ; set PASS_THROUGH modality
            whatTemp[rid] = 3 ; next step
            ; break intentionally omitted
        case 3
            ero = co_robj(sobj[rid])
            select(ero)
                default
                    whatTemp[rid] = 0 ; go back
                    return ero ; return error code
                case 0
                    return 1 ; we have to wait
                case -2 ; busy, restart again next loop
                    sObj[rid].phase = 1 ; set PASS_THROUGH modality
for the next call
                    return 1
                case 1 ; we got the answer and we save it
                    Heater_temp[rid] = sObj[rid].data
                    break
            end_select
            sObj[rid].objCode = 0x2101 ; Control Board Temp
            sObj[rid].subIndex = 0xC
            sObj[rid].type = 5 ; float data
            sObj[rid].phase = 1 ; set PASS_THROUGH modality
            whatTemp[rid] = 4 ; next step
            ; break intentionally omitted
        case 4
            ero = co_robj(sobj[rid])
            select(ero)
                default
                    whatTemp[rid] = 0 ; go back
                    return ero ; return error code
                case 0
                    return 1 ; we have to wait
                case -2 ; busy, restart again next loop
                    sObj[rid].phase = 1 ; set PASS_THROUGH modality
for the next call
                    return 1
                case 1 ; we got the answer and we save it
                    ControlBoard_temp[rid] = sObj[rid].data
                    break
            end_select
            whatTemp[rid] = 0 ; end of the job ....
            break
    end_select
```

© 2025 Robox SpA

```
    return 2 ; all temperature updated
ENDFUN

;<0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0>
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
FUNCTION int ReadImdTempBL(int rid)
    ; routine 'BLOCKING' version
    ; function to update the IMD temperature value each PER_UPD_TEMP
seconds.
    ; the information will be set in global storage.
    ; the function want the RID of the IMD
    ; give back:
    ; <0 error
    ; >2 error (SDO diagnostic)
    ; 0 no operation
    ; 2 all temperature updated
    int ero, timeToRead
    stru_co_obj sobj
    ; check RID value
    if(not range(rid, 1, 32))
        return -1 ; return error if illegal value
    endif
    timeToRead = FALSE
    ; check if time to update
    if(updateTime[rid] <= tfb)
        inc(updateTime[rid], PER_UPD_TEMP)
        timeToRead = TRUE
    endif
    ; check what we have to do
    if(not timeToRead)
        return 0 ; nothing to do
    endif
    ; we can read
    sObj.rid = rid
    sObj.objCode = 0x2101 ; Cpu drive Temp
    sObj.subIndex = 2
    sObj.type = 5 ; float data
    sObj.phase = -1 ; set Blocking modality
    ero = co_robj(sobj)
    if(ero <> 1)
        return ero
    endif
    CpuDrive_temp[rid] = sObj.data
    ; prepare the read next temperature
    sObj.subIndex = 1 ; Heater Temp
    sObj.phase = -1 ; set PASS_THROUGH modality
    ero = co_robj(sobj)
    if(ero <> 1)
        return ero
```

© 2025 Robox SpA

```
    endif
    Heater_temp[rid] = sObj.data
    ; prepare the read next temperature
    sObj.subIndex = 0xC ; Control Board Temp
    sObj.phase = -1 ; set PASS_THROUGH modality
    ero = co_robj(sobj)
    if(ero <> 1)
        return ero
    endif
    ControlBoard_temp[rid] = sObj.data
    return 2 ; all temperatures updated
ENDFUN
```

Function co_rpdo()

This function is used to read the data in the buffer (PDO - Process Data Object) received from the indicated workstation. It is also used to send the request for pdo update (RTR).

Syntax	**i32 co_rpdo (i32 nWs, i32 nPdo, i32 nEl, real where)**
nWs	Robox station ID (logical workstation 1-:-64). **Note**: If 0 is specified as ws then it is meant for all.
nPdo	Number of the PDO being received from the drive (1-:-4)
nEl	Number of the element that you want to read: • 0, rPDO data refresh request command (RTR for asynchronous PDO) • 1-:-8, number of the logical element to be read **NOTE**: If *nEl* is 0, the value is not returned.
where	Parameter in which the value read by PDO is downloaded. Conversion from the right format (defined in the configuration file) to real is performed by the base sw
Result	Returns:

Condition	Value	Description
-	0	Illegal parameters
nEl = 1	1	Data read for the first time. **Note**: Meaningful

© 2025 Robox SpA

Syntax	**i32 co_rpdo (i32 nWs, i32 nPdo, i32 nEl, real where)**
	<table><tr><th>Condition</th><th>Value</th><th>Description</th></tr><tr><td></td><td></td><td>information only if this instruction is executed in the rules to all scans.</td></tr><tr><td>*nEl* = 1</td><td>2</td><td>Data already read. **Note**: Significant information only if this instruction is executed in the rules at all scans.</td></tr><tr><td>*nEl* <> 1</td><td>3</td><td>Valid data</td></tr></table>
Validity	Rule, Task
Note	

Function co_send_sdo()

WARNING: This function cannot be used in a task if the co_robj 249 ()/co_wobj()/coe_robj 255 ()/coe_wobj 259 () functions are also present in the task.

This function is used to send an SDO (Service Data Object) message to the specified workstation in order to write/read an object from the object dictionary.

- In case of write -> co_send_sdo(*nWs*, *nObj*, *nSub*): the co_asw_sdo 215 () function is then used to check if the write was successful
- In case of read -> co_send_sdo(*nWs*, *nObj*, *nSub*, *type*, *data*): the co_asw_sdo() function is then used to check if the read request was successful and to download the data

Syntax	**i32 co_send_sdo (i32 nWs, i32 nObj, i32 nSub)** **i32 co_send_sdo (i32 nWs, i32 nObj, i32 nSub, i32 type, real data)**
nWs	Robox station ID (logical workstation 1-:-64)
nObj	Dictionary object code (see dictionary - eds file)
nSub	Sub index code (see dictionary - eds file). In case of EtherCAT (SoE) see below

Syntax	**i32 co_send_sdo (i32 nWs, i32 nObj, i32 nSub)** **i32 co_send_sdo (i32 nWs, i32 nObj, i32 nSub, i32 type, real data)**
type	Allows you to select how to transform the data to be written contained in *data* into the type requested by the connected device. Value / Type 1 / Signed/unsigned 8 bit. 2 / Signed/unsigned 16 bits. 3 / Signed/unsigned 24 bits. 4 / Signed/unsigned 32 bits. 5 / Real 32 bit.
data	Value to transform (from real to FIX format of size specified in *type*)
Result	Returns: Value / Description >= 0 / Message handle number. (To be used later as an incoming parameter in the co_asw_sdo 215() function) -1 / Illegal parameters -2 / RTE message handler busy
Validity	Rule, Task
Note	Avoid, if the statement is used within a function, declaring local handles to the function itself. From RTE 34.13.5, the functions co_robj 234() and co_wobj 249() are available

Ethercat SoE sub index parameter

Bit	Hex value	Meaning
0	0x00000001	Value-coded:

© 2025 Robox SpA

Bit	Hex value	Meaning
1	0x00000002	• 0 if single-device • Index of the one concerned in case of multidevice
2	0x00000004	
3	0x00000008	Reserved (= 0)
4	0x00000010	Reserved (= 0)
On writing operation:		
5	0x00000020	In 1 request for abort of command associated with specified IDN. **NOTE**: command end code is still sent.
6	0x00000040	In 1 enable status notification management of command associated with specified IDN (if associated with start code) (value 3)
7	0x00000080	In 1 enable automatic command handling (with command start send and on command end execution notification send end code). If the command lasts longer than the timeout time by a few seconds, the response wait routine will timeout. The SoE command will still remain active
On read operation:		
5	0x00000020	Coded to value: • -2 command aborted or never started • 0 command running • 3 command executed successfully • 15 error during command execution
6	0x00000040	
7	0x00000080	

Examples

Example of use:

```
; if there is a reading command
if(rise(command_read))
    command_read = false
```

```
    faseRead = SEND
endif
select(faseRead)
    case IDLE
    break
    case SEND
        handle = co_send_sdo 242 (rNws, rObj, rSub)
        if(handle < 0)
            faseRead = ERROR
            break
        endif
        faseRead = GET_DATA
    break
    case GET_DATA
       result = co_asw_sdo 215 (rNws, handle, rType, Data)
        ; waiting answer
        _if(result =0)
            break
        ; test answer
        if(result = 1)
            faseRead = DONE
        else
            faseRead = ERROR
        endif
    break
    case DONE
        ; insert in the MREPORT diagnosys from SHELL
        str_format 601 (srInfo, "SDO Read ws:%d, obj: %d, sub:%d,
returned value:%f", rnWs, rObj, rSub, Data)
        inf_report 697 (0x1f, srInfo)
        faseRead = IDLE
    break
    case ERROR
        ; insert in the MREPORT diagnosys from SHELL
        str_format(srInfo, "Error SDO read value:%d", result)
        inf_report(0x1f,srInfo)
        faseRead = IDLE
    break
end_select

; WRITE
INT handle, returned
; write object 2000 sub 0x92 of workstation 1.
; (e.g., on HID corresponds to pr 145 -Kp position ring-) to value
100 decimal (64 H)
handle = co_send_sdo(1, 0x2000, 0x92, 4, 0x64)

returned = co_asw_sdo(1, handle)
_if(returned = 1)
 sr(1) = "successful write"
_if(returned = 0)
 sr(1) = "Message waiting for response"
```

© 2025 Robox SpA

```
_if(returned < 0)
 sr(1) = "errors"

; READ
INT handle, returned
REAL data
; read object 2000 sub 0x92 from workstation 1
handle = co_send_sdo(1, 0x2000, 0x92)
_if(co_asw_sdo(1, handle, 2, data) = 1)
 ; test for response arrival and data download
 ; ...
```

Function co_statusword()

Function that returns the contents of the indicated workstation status word.

Note: From RTE 34.03.06, the keywords fb_stsw and fb_ctrlw are available.

Syntax	**i32 co_statusword (i32 nWs [, i32 flgCw=FALSE])**
nWs	Robox station ID (logical workstation 1-:-64). **Note**: If 0 is specified as ws then it is meant for all.
flgCw	Parameter to also enable reading the control word as well. (optional, default FALSE)
Result	Returns:

Condition	Value	Description
flgCw = 0	Value of status word	Value returned in w0 644 (lower part, first word)
	0	Value returned in w1 (upper part, second word)
flgCw <> 0	Value of the status word	Value returned in w0
	Value of the control word	Value returned in w1

Validity	Rule, Task
Note	

Status word bit

© 2025 Robox SpA

Bit	Hexadecimal value	Description
0	0x00000001	Ready to switch ON
1	0x00000002	Switched ON
2	0x00000004	Operation enabled
3	0x00000008	Fault
4	0x00000010	Voltage enabled
5	0x00000020	Quick stop
6	0x00000040	Switch on disabled
7	0x00000080	Warning
8	0x00000100	Manufacturer specific
9	0x00000200	Remote
10	0x00000400	Target reached
11	0x00000800	Internal limit active
12	0x00001000	Operation mode specific
13	0x00002000	Operation mode specific
14	0x00004000	Manufacturer specific
15	0x00008000	Manufacturer specific

The figure was extrapolated from CiA Draft Standard Proposal 402.

Examples

Example of use:

```
int stws[33] ; stsws[0] is not used
for(i= 1 , i<=32 , i++)
    stws[i] = co_statusword 246 (i)
end_for
```

Example of use:

```
reso = co_statusword(ws,true)
r(1000) = w0 644 (reso)    ; status word
r(1001) = w1 644 (reso)    ; control word
```

Function co_sync()

WARNING: Obsolete.

This function is used to enable/disable SYNC message transmission on all CANopen DS402 interfaces (SW CODMnnvv.xxx) in the system.

© 2025 Robox SpA

Syntax	**i32 co_sync (i32 flag)**
flag	Parameter to enable/disable sending of SYNC message: • <> 0, enable sending (defaults to power-on for RTE manager) • 0, disable sending
Result	Returns: Value: Description 0: CANopen not present 1: Successful enablement
Validity	Rule, Task
Note	Enablement is handled by RTE

Function co_wnmt()

Function that imposes the NMT operating mode of the indicated workstation.

Syntax	**i32 co_wnmt(i32 nWs, i32 opeMode)**
nWs	Robox station ID (logical workstation 1-:-64). **Note**: If 0 is specified as ws then it is meant for all.
opeMode	Imposable operating mode:

Value	Code	Mode
1	0x01	ENTER OPERATIONAL
2	0x02	ENTER STOP
128	0x80	ENTER PREOPERATIONAL
129	0x81	RESET NODE
130	0x82	RESET COMMUNICATION

Result	Returns:

Value	Description
<> 0	Successful write

© 2025 Robox SpA

Syntax	**i32 co_wnmt(i32 nWs, i32 opeMode)**
	Value: 0 — Description: Illegal parameters encountered
Validity	Rule, Task
Note	Action can also be obtained from shell command

Function co_wobj()

WARNING: This function cannot be used in a task if the co_send_sdo 242() / co_asw_sdo 215() functions are also present in the task.

This function is used to perform writing of fieldbus objects. Writing can be done in pass-through mode or in blocking mode (blocking mode can be used only in tasks 1-:-8).

Pass-through mode means that the function must be continuously called and this returns the current state of the operation.

Blocking mode means that the function is invoked and returns only when it has completed the write operation.

The function returns a result on the state of execution.

Syntax	**i32 co_wobj (stru_co_obj stru)**
stru	Structure name of type stru_co_obj 805
Result	Variable of type i32 containing the result. See pass-through mode and/or blocking mode
Validity	Rule, Task
Note	

Pass-through mode

- Set ***stru*.phase = 1** to initialize the write
- The write command is sent to the remote device and a response is expected. The variable *stru*.phase is changed.
  The function returns 0
- Waiting for response.
  The function continues to return 0
- Receiving response with **positive** result. The value read is written to *stru*.data. RTE changes *stru*.phase = 4.
  Function returns 1
- Receive response with **negative** result. RTE changes *stru*.phase = 0.
  Function returns error code:
  - -1, illegal parameters(alarm 9900)

© 2025 Robox SpA

- -2, communication handler busy
- -3, communication handler crash (restart)
- -5, command to a device not present(*stru*.rid)
- -6, no operation running. *stru*.phase not initialized
- 2, command already received positive response, value available in *stru*.data
- >2, SDO error code, see table below

Blocking mode

- Set ***stru*.phase = -1** to execute the write
- The write command is sent to the remote device and a response is expected. The variable *stru*.phase is changed
- Waiting for response. If the response does not arrive within a maximum of 3 seconds, the function exits with an error (timeout, returns -2)
- Receiving response with a **positive** result. The value read is written to *stru*.data. RTE changes *stru*.phase = 4.
  Function returns 1
- Receive response with **negative** result. RTE changes *stru*.phase = 0.
  Function returns error code:
  - -1, illegal parameters (alarm 9900)
  - -2, communication handler busy
  - -3, communication handler crash (restart)
  - -4, blocking mode handling not allowed in current task(*stru*.phase = 1) (rule, ladder). Command not accepted
  - -5, command to a device not present(*stru*.rid)
  - -6, no task running. *stru*.phase not initialized
  - 2, command already responded to positively, value available in *stru*.data
  - >2, SDO error code, see table below

SDO SoE/CoE Error Codes

SDO error code	Description
0x0503 0000	Toggle bit not toggled
0x0504 0000	SDO protocol timeout
0x0504 0001	Invalid or unknown client/server command identifier
0x0504 0002	Invalid block size (blocking mode only)
0x0504 0003	Invalid sequence number (blocking mode only)

© 2025 Robox SpA

SDO error code	Description
0x0504 0004	CRC error (blocking mode management only)
0x0504 0005	Memory overflow
0x0601 0000	Unsupported access to an object
0x0601 0001	Attempt to read a write-only object
0x0601 0002	Attempt to write a read-only object
0x0602 0000	Object does not exist in the object dictionary
0x0604 0041	Unable to map the object to the PDO
0x0604 0042	The number and length of objects to be mapped would exceed the length of the PDO
0x0604 0043	Incompatibility of parameters
0x0604 0047	Incompatibility internal to the device
0x0606 0000	Failed access due to hardware error
0x0607 0010	Data type does not match, length of service parameter does not match
0x0607 0012	Data type does not match, length of service parameter is too high
0x0607 0013	Data type does not match, service duration parameter is too low
0x0609 0011	Subindex does not exist
0x0609 0030	Field of parameter values exceeded (write access only)
0x0609 0031	Parameter value written too high
0x0609 0032	Value of the written parameter too low
0x0609 0036	Maximum value is less than minimum value
0x0800 0000	General error
0x0800 0020	Data cannot be transferred or stored in the application
0x0800 0021	Data cannot be transferred or stored in the application due to local control
0x0800 0022	Data cannot be transferred or stored in the application due to the current state of the device

© 2025 Robox SpA

SDO error code	Description
0x0800 0023	Dynamic object dictionary generation fails or no object dictionary is present (e.g., object dictionary is generated from a file and generation fails due to a file error)

Examples

```
; SAMPLE CODE
; Global Storage definition
;---------------------------

stru_co_obj 805 swobj[32+1]
int resW
....
....
....
; version using pass-through routine
if(nm <> 0)
    do
        resW = SetImdTorqueMaxPT(2, nm)
        ...
        ...
    end_do_while(resW = 1)
    nm = 0
endif
....
....
....
; version using blocking routine
if(nm <> 0)
    resW = SetImdTorqueMaxBL (2, nm)
    nm = 0
endif
....
....
....

;<0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0>
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
FUNCTION int SetImdTorqueMaxPT(int rid, real nm)

    ; routine 'PASS_THROUGH' version
    ; function to update the MAX_TORQUE on Imd.
    ; the function want the RID of the IMD, and the value in NM to
set
    ; it gives back:
    ; <0 error
    ; >2 error (SDO diagnostic)
```

```
    ; 0 no operation
    ; 1 operation in progress
    ; 2 torque has been written
    int ero
    ; check RID value
    if(not range(rid, 1, 32))
        return -1 ; return error if illegal value
    endif
    if((swObj[rid].phase = 0 ) or (swobj[rid].phase = 4) )
        swobj[rid].rid = rid
        swObj[rid].objCode = 0x2050 ; Absolute Torque Limit
        swObj[rid].subIndex = 9
        swObj[rid].type = 5 ; float data
        swObj[rid].data = nm ; value to write
        swObj[rid].phase = 1 ; set PASS_THROUGH modality
    endif
    ; check what we have to do
    ero = co_wobj(swobj[rid])
    select(ero)
        default
            whatTemp[rid] = 0 ; go back
            return ero ; return error code
        case 0
            return 1 ; we have to wait
        case -2 ; busy, restart again next loop
            sObj[rid].phase = 1 ; set PASS_THROUGH modality for the
next call
            return 1
        case 1 ; we got the answer that the parameters has been
written
            break
    end_select
    return 2 ; the parameters has been written
endfun

;<0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0>
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
FUNCTION int SetImdTorqueMaxBl(int rid, real nm)

    ; routine 'BLOCKING' version
    ; function to update the MAX_TORQUE on Imd.
    ; the function want the RID of the IMD, and the value in NM to
set
    ; it gives back:
    ; <0 error
    ; >2 error (SDO diagnostic)
    ; 0 no operation
    ; 1 operation in progress
    ; 2 torque has been written
    int ero
```

© 2025 Robox SpA

```
    stru_co_obj swobj
    swobj.rid = rid
    swObj.objCode = 0x2050 ; Absolute Torque Limit
    swObj.subIndex = 9
    swObj.type = 5 ; float data
    swObj.data = nm ; value to write
    swObj.phase = -1 ; set BLOCKING modality
    ; check what we have to do
    ero = co_wobj(swobj)
    if(ero <> 1)
        return ero ; error
    endif
    return 2 ; the parameters has been written
ENDFUN
```

Function co_wpdo()

This function is used to write data into the transmission buffer (PDO - Process Data Object) to the indicated workstation.

Syntax	**i32 co_wpdo (i32 nWs, i32 nPdo, i32 nEl, real value)**
nWs	Robox station ID (logical workstation 1-:-64). **Note**: If 0 is specified as ws then it is meant for all.
nPdo	Number of the PDO being transmitted to the drive (1-:-4)
nEl	Number of the element that you want to write: • 0, transmit command (for asynchronous PDO) • 1-:-8, number of the logical element within the PDO **Note**: If *nEl* is 0, the value does not count.
value	Value to be entered into the PDO. Conversion to the correct format (defined in the configuration file) is performed by the base sw
Result	Returns: Value: <> 0 — Successful write Value: 0 — Illegal parameters detected

© 2025 Robox SpA

Syntax	**i32 co_wpdo (i32 nWs, i32 nPdo, i32 nEl, real value)**
Validity	Rule, Task
Note	

Function coe_robj()

WARNING: This function cannot be used in a task if the co_send_sdo 242()/co_asw_sdo 215() functions are also present in the task.

This function is used to read CoE fieldbus objects (the object defined by '*stru*.objCode + *stru*.subIndex' on the device identified by *stru*.rid is read).

It can read more data types than the analogous function co_robj 234().

Reading can be done in pass-through mode or in blocking mode (usable only in tasks 1-:-8).

Pass-through mode means that the function must be continuously called and simultaneously tested the current state of the task.

Blocking mode means that the function is invoked and returns only when it has completed the read operation.

The function returns a result on the state of execution.

Syntax	**i32 co_robj (stru_coe_obj stru, destination)**
stru	Structure name of type stru_coe_obj 807
destination	Contains the information received. *destination* must be an object compatible with the data being read (if reading a VISIBLE STRING, *destination* will be of type string, if reading an OCTET_STRING it will be a byte array or structure)
Result	Variable of type i32 containing the result. See pass-through mode and/or blocking mode
Validity	Rule, Task
Note	

Pass-through mode

- Set ***stru*.phase = 1** to initialize the read
- The read command is sent to the remote device and a response is expected. The variable *stru*.phase is changed.
  The function returns 0
- Waiting for response.
  The function continues to return 0
- Response is received with a **positive** result. The value read is written to *destination*.
  RTE changes *stru*.phase = 4.
  Function returns 1

© 2025 Robox SpA

- Receive response with **negative** result. RTE changes *stru*.phase = 0.
  Function returns error code:
  - -2, communication handler busy
  - -3, communication handler crash (restart)
  - -5, command to a device not present(*stru*.rid)
  - -6, no operation running. *stru*.phase not initialized
  - -12, illegal value of objcode(alarm 9900 arg 12)
  - -13, illegal value of subindex (alarm 9900 arg 13)
  - -14, illegal value of datatype (alarm 9900 arg 4/arg 14)
  - -15, illegal value of datalen (alarm 9900 arg 5/arg 15)
  - -16, illegal value of phase (alarm 9900 arg 6)
  - -21, destination not specified (alarm 9900 arg 16)
  - -22, destination changed (alarm 9900 arg 32)
  - 2, command for which a positive response has already been received, value available in *destination*
  - >2, SDO error code, see table below

Blocking mode

- Set ***stru*.phase = -1** to perform the read
- The read command is sent to the remote device and a response is expected. The variable *stru*.phase is changed
- Waiting for response. If the response does not arrive within a maximum of 3 seconds, the function exits with an error (timeout, returns -2)
- Receiving response with a **positive** result. The value read is written to *destination*. RTE changes *stru*.phase = 4.
  Function returns 1
- Receive response with **negative** result. RTE changes *stru*.phase = 0.
  Function returns error code:
  - -2, communication handler busy
  - -3, communication handler crash (restart)
  - -4, blocking mode not allowed in current task(*stru*.phase = 1) (rule, ladder).
    Command not accepted
  - -5, command to a device not present(*stru*.rid)
  - -6, no task running. *stru*.phase not initialized
  - -12, illegal value of objcode(alarm 9900 arg 12)
  - -13, illegal value of subindex (alarm 9900 arg 13)
  - -14, illegal value of datatype (alarm 9900 arg 4/arg 14)
  - -15, illegal value of datalen (alarm 9900 arg 5/arg 15)
  - -16, illegal value of phase (alarm 9900 arg 6)
  - -21, destination not specified (alarm 9900 arg 16)

© 2025 Robox SpA

- -22, destination changed (alarm 9900 arg 32)
- 2, command for which a positive response has already been received, value available in *destination*
- >2, SDO error code, see table below

SDO SoE/CoE Error Codes

SDO error code	Description
0x0503 0000	Toggle bit not toggled
0x0504 0000	SDO protocol timeout
0x0504 0001	Invalid or unknown client/server command identifier
0x0504 0002	Invalid block size (blocking mode only)
0x0504 0003	Invalid sequence number (blocking mode only)
0x0504 0004	CRC error (blocking mode management only)
0x0504 0005	Memory overflow
0x0601 0000	Unsupported access to an object
0x0601 0001	Attempt to read a write-only object
0x0601 0002	Attempt to write a read-only object
0x0602 0000	Object does not exist in the object dictionary
0x0604 0041	Unable to map the object to the PDO
0x0604 0042	The number and length of objects to be mapped would exceed the length of the PDO
0x0604 0043	Incompatibility of parameters
0x0604 0047	Incompatibility internal to the device
0x0606 0000	Failed access due to hardware error
0x0607 0010	Data type does not match, length of service parameter does not match
0x0607 0012	Data type does not match, length of service parameter is too high
0x0607 0013	Data type does not match, service duration parameter is too low
0x0609 0011	Subindex does not exist

© 2025 Robox SpA

SDO error code	Description
0x0609 0030	Field of parameter values exceeded (write access only)
0x0609 0031	Parameter value written too high
0x0609 0032	Value of the written parameter too low
0x0609 0036	Maximum value is less than minimum value
0x0800 0000	General error
0x0800 0020	Data cannot be transferred or stored in the application
0x0800 0021	Data cannot be transferred or stored in the application due to local control
0x0800 0022	Data cannot be transferred or stored in the application due to the current state of the device
0x0800 0023	Dynamic object dictionary generation fails or no object dictionary is present (e.g., object dictionary is generated from a file and generation fails due to a file error)

Examples

Example of use:

```
; SAMPLE CODE
; =======
; in task
; =======

stru_coe_obj 807 rCoe
int          esitoFB
string       rCoeS
u8           rCoeO[256]
...
...
    rCoe.rid      = 2        ; axioline
    rCoe.objCode  = 0x9002   ; module specific information
    rCoe.subIndex = 1        ; product name
    rCoe.datatype = 9        ; visible string
    rCoe.datalen  = 128      ; dimension max string
    rCoe.phase    = -1
    esitoFB = coe_robj 255(rCoe, rCoeS)
    if(esitoFB < 0)
        ; error
        ...
    else
```

```
        ; now the string can be used
         ....
    endif
    ...
    ...
    ...
    rCoe.rid      = 2        ; axioline
    rCoe.objCode  = 0x9002   ; module specific information
    rCoe.subIndex = 0xb      ; Device Type
    rCoe.datatype = 10       ; octet
    rCoe.datalen  = 256
    rCoe.phase    = -1
    esitoFB = coe_robj(rCoe, rCoeO)
    if(esitoFB < 0)
        ; error
         ...
    else
        ; now the data of the array can be used
         ....
    endif
```

Function coe_wobj()

WARNING: This function cannot be used in a task if the co_send_sdo 242 ()/co_asw_sdo 215 () functions are also present in the task.

This function is used to perform writing of CoE fieldbus objects (information present in *source* is written).

It can write more data types than the analogous function co_wobj 249 ().

Writing can be done in pass-through mode or in blocking mode (usable only in tasks 1-:-8).

Pass-through mode means that the function must be continuously called and simultaneously tested the current state of the task.

Blocking mode means that the function is invoked and returns only when it has completed the write operation.

The function returns a result on the state of execution.

Syntax	**i32 coe_wobj (stru_coe_obj stru, source)**
stru	Structure name of type stru_coe_obj 807
source	Data contained in *source* will be taken according to the specified data type (if writing a VISIBLE STRING, *source* will be of type string, if writing an OCTET_STRING, *source* will be a byte array or structure)
Result	Variable of type i32 containing the result. See pass-through mode and/or blocking mode
Validity	Rule, Task

© 2025 Robox SpA

Syntax	**i32 coe_wobj (stru_coe_obj stru, source)**
Note	

Pass-through mode

- Set ***stru*.phase = 1** to initialize the write
- The command to write the *source* value is sent to the remote device and a response is expected. The variable *stru*.phase is changed.
  The function returns 0
- Waiting for response.
  The function continues to return 0
- Receiving response with **positive** result. The value read is written to *source*. RTE changes *stru*.phase = 4.
  Function returns 1
- Receive response with **negative** result. RTE changes *stru*.phase = 0.
  Function returns error code:
  - -2, communication handler busy
  - -3, communication handler crash (restart)
  - -5, command to a device not present(*stru*.rid)
  - -6, no operation running. *stru*.phase not initialized
  - -12, illegal value of objcode(alarm 9900 arg 12)
  - -13, illegal value of subindex (alarm 9900 arg 13)
  - -14, illegal value of datatype (alarm 9900 arg 4/arg 14)
  - -15, illegal value of datalen (alarm 9900 arg 5/arg 15)
  - -16, illegal value of phase (alarm 9900 arg 6)
  - -21, *source* not specified (NULL) (alarm 9900 arg 16)
  - -22, *source* changed (phase execution) (alarm 9900 arg 32)
  - 2, command for which a positive response has already been received, value available in *source*
  - >2, SDO error code, see table below

Blocking mode

- Set ***stru*.phase = -1** to execute the write
- The command to write the *source* value is sent to the remote device and a response is expected. The variable *stru*.phase is changed
- Waiting for response. If the response does not arrive within a maximum of 3 seconds, the function exits with an error (timeout, returns -2)
- Receiving response with a **positive** result. The value read is written to *source*. RTE changes *stru*.phase = 4.
  Function returns 1
- Receive response with **negative** result. RTE changes *stru*.phase = 0.
  Function returns error code:

© 2025 Robox SpA

- -2 communication handler busy
- -3 communication handler crash (restart)
- -4 blocking mode not allowed in current task(*stru*.phase = 1) (rule, ladder). Command not accepted
- -5 command to a device not present(*stru*.rid)
- -6 no task running. *stru*.phase not initialized
- -12 illegal objcode value(alarm 9900 arg 12)
- -13 illegal value of subindex (alarm 9900 arg 13)
- -14 illegal value of datatype (alarm 9900 arg 4/arg 14)
- -15 illegal value of datalen (alarm 9900 arg 5/arg 15)
- -16 illegal phase value (alarm 9900 arg 6)
- -21 *source* not specified (NULL) (alarm 9900 arg 16)
- -22 *source* changed (phase execution) (alarm 9900 arg 32)
- 2 command for which a positive response has already been received, value available in *source*
- >2 SDO error code, see table below

SDO SoE/CoE Error Codes

SDO error code	Description
0x0503 0000	Toggle bit not toggled
0x0504 0000	SDO protocol timeout
0x0504 0001	Invalid or unknown client/server command identifier
0x0504 0002	Invalid block size (blocking mode only)
0x0504 0003	Invalid sequence number (blocking mode only)
0x0504 0004	CRC error (blocking mode management only)
0x0504 0005	Memory overflow
0x0601 0000	Unsupported access to an object
0x0601 0001	Attempt to read a write-only object
0x0601 0002	Attempt to write a read-only object
0x0602 0000	Object does not exist in the object dictionary
0x0604 0041	Unable to map the object to the PDO

© 2025 Robox SpA

SDO error code	Description
0x0604 0042	The number and length of objects to be mapped would exceed the length of the PDO
0x0604 0043	Incompatibility of parameters
0x0604 0047	Incompatibility internal to the device
0x0606 0000	Failed access due to hardware error
0x0607 0010	Data type does not match, length of service parameter does not match
0x0607 0012	Data type does not match, length of service parameter is too high
0x0607 0013	Data type does not match, service duration parameter is too low
0x0609 0011	Subindex does not exist
0x0609 0030	Field of parameter values exceeded (write access only)
0x0609 0031	Parameter value written too high
0x0609 0032	Value of the written parameter too low
0x0609 0036	Maximum value is less than minimum value
0x0800 0000	General error
0x0800 0020	Data cannot be transferred or stored in the application
0x0800 0021	Data cannot be transferred or stored in the application due to local control
0x0800 0022	Data cannot be transferred or stored in the application due to the current state of the device
0x0800 0023	Dynamic object dictionary generation fails or no object dictionary is present (e.g., object dictionary is generated from a file and generation fails due to a file error)

Examples

Example of use:

```
; SAMPLE CODE
; =======
; in task
; =======
```

```
struct_p tunnelCommnad
  u8 u
  u8 pdiH
  u8 pdiL
  u8 sidx
end_struct_p
stru_coe_obj 807     wCoe, rCoe
int              esitoFB
TunnelCommnad    tnRead
u8               rCoeO[256]
...
...
    wCoe.rid      = 2        ; axioline
    wCoe.objCode  = 0x3000   ; Axioline F module nn PDI read tunnel
    wCoe.subIndex = 1        ; command
    wCoe.datatype = 10
    wCoe.datalen  = sizeof(tnRead)   ; size of data to write
    wCoe.phase    = -1

    tnRead.u      = 0
    tnRead.pdiH   = 0
    tnRead.pdiL   = 0x18
    tnRead.sidx   = 0
    rCoe.rid      = 2        ; axioline
    rCoe.objCode  = 0x3000   ; Axioline F module nn PDI read tunnel
    rCoe.subIndex = 0x3      ; Response
    rCoe.datatype = 10       ; octet
    rCoe.datalen  = 256
    rCoe.phase    = -1
    esitoFB = coe_wobj 259(wCoe, tnRead)
    if(esitofb < 0)
       ; error on writing
        ....
    endif

    ; successful write command
    ; to complete the tunneling of the PHOENIX- AXIOLINE product
    ; let's go read the answer...
    esitoFB = coe_robj(rCoe, rCoeO)
    if(esitofb < 0)
       ; error
        ....
    endif

    ; I now have the answer in rCoeO of the pdi 18 info
    ...
    ...
```

© 2025 Robox SpA

Function command()

Function that specifies an ascii directive to be executed by axis control.

Syntax	**string command (string strCmd)**
strCmd	Parameter containing the directive to be executed by the motion control
Result	Returns:

Value	Description
Directive response	The directive has been recognized and a response is expected
"unknown command"	The directive has not been recognized
"OK"	The directive has been executed and a response is not expected

Validity	Rule, Task
Note	

The chapter 'connectivity-->free protocol serial communication' illustrates the possibility, via serial connection, of sending properly formatted serial commands and obtaining responses from RTE.

This function is precisely to "simulate" the request made by serial.

Examples

Example of use:

```
sr(4) = command(264)("D R 1")
; sr(4) = "    0 0x00000000")
;-----------------------------
sr(4) = command("D AL 1")
; sr(4) = " 0 no alarm"
;-----------------------------
sr(4) = command("sysinfo")
; sr(4) = "unknown command"
;-----------------------------
sr(4) = command("f_inp 171")
; sr(4) = "ok"
```

Function cond_val()

Function that allows a unit to be imposed as a function of the logical state of a condition.

Syntax	**real cond_val (cond, real valTrue, real valFalse)**
cond	Condition to be examined. It can also be a generic expression
valTrue	Result value in case of *cond* TRUE
valFalse	Result value in case of *cond* FALSE
Result	Returns *valTrue* or *valFalse* based on the logical state of the condition
Validity	Rule, Task
Note	

Examples

Example of use:

```
rr(1) = cond_val 265(r(100) = 4, 24.18, -24.18)
; equivalent to
if(r(100) = 4))
    rr(1)= 24.18
else
    rr(1)= -24.18
endif

;--------
int x = 5
int y = 8
int result = cond_val(x > y, inc 351(x), inc(y))
; At this point x = 5 e y = 9
```

Function cos()

Function that performs the cosine of the specified angle in radians.

Syntax	**real cos (real value)**
value	Cosine argument. **Note**: *value* must be in radians.
Result	Returns the cosine of *value*
Validity	Rule, Task
Note	

Examples

© 2025 Robox SpA

Example of use:

```
rr(5) = 0

rr(10) = cos(rr(5))              ; = 1
rr(10) = cos(K_PI/4)             ; = 0.7071...
rr(101) = cos(torad(rr(100))) ; rr (100) --> angle in degrees
                              ; rr (101) --> cosine of the angle
contained in rr(100)
```

Function crc16_be()

Given a data buffer (byte sequence), this function performs the CRC16 calculation based on the specified polynomial and initial CRC value.

Syntax	**u16 crc16_be (string strData, i32 nCount [, u16 initVal=0xFFFF] [, u16 polin=0xA001])**
strData	String or array, of which CRC16 is to be calculated. However, the CRC calculation is done on bytes, regardless of the type specified for the given variable
nCount	This is the number of bytes of *strData* on which to perform the calculation
initVal	Initial value of CRC. (optional, default 0xFFFF)
polin	Value of the polynomial used in CRC calculation. (optional, default 0xA001) Main polynomials used by different CRCs:

Polynomial	*Polin* value
Normal original	0x8005
Normal reversed	0xA001
Koopman original	0xC002
Normal reciprocal	0x4003
Reversed reciprocal	0xC002
Koopman reciprocal	0xA001

Frequent cases:

Most used CRCs	Representation	NORMAL	REVERSE	RECIPROCAL	Used by
CRC-16-CCITT	x16 + x12 + x5 + 1	0x1021	0x8408	0x0811	Modem X25,

Syntax	**u16 crc16_be (string strData, i32 nCount [, u16 initVal=0xFFFF] [, u16 polin=0xA001])**

Most used CRCs	Representation	NORMAL	REVERSE	RECIPROCAL	Used by
					Bluetooth, PPP
CRC-16-IBM	x16 + x15 + x2 + 1	0x8005	0xa001	0x4003	USB, MODBUS, altri

Result	Returns the calculated CRC value
Validity	Rule, Task
Note	By repeatedly calling the function with initial value equal to the calculated CRC value, it is possible to perform incremental calculation of the CRC of a buffer of any length. **NOTE**: The previous functions crc16 and crc16_init are deprecated, but remain for compatibility.

The calculation is done according to the Big Endian algorithm.

It differs from the crc16_le 268() algorithm because a shift of the CRC being processed is made to the left instead of the right and because the data byte is inserted at the top instead of the bottom of the word on which the calculation is performed.

Examples

Example of use:

```
crcData = crc16_be 266(data, numBytes, crcInit, pol)
```

Valore crcData	data *(strData)*	numBytes *(nCount)*	crcInit *(initVal)*	pol *(polin)*
50244	"d ve"	4	0xFFFF	0xA001
16302	"d ve"	4	0	0xA001
35609	"d ve"	4	0	0x1021
1978	"13564"	5	0xFFFF	0xA001
11532	"13564"	5	0	0xA001

© 2025 Robox SpA

Function crc16_le()

This function, given a data buffer (sequence of bytes), performs the CRC16 calculation based on the specified polynomial and initial CRC value.

Syntax	**u16 crc16_le (string strData, i32 nCount [, u16 initVal=0xFFFF] [, u16 polin=0xA001])**
strData	String or array of which CRC16 is to be calculated. However, the CRC calculation is done on bytes, regardless of the type specified for the given variable
nCount	This is the number of bytes of *strData* on which to perform the calculation
initVal	Initial value of CRC. (optional, default 0xFFFF)
polin	Value of the polynomial used in CRC calculation. (optional, default 0xA001) Main polynomials used by different CRCs:

Polynomial	*Polin* value
Normal original	0x8005
Normal reversed	0xA001
Koopman original	0xC002
Normal reciprocal	0x4003
Reversed reciprocal	0xC002
Koopman reciprocal	0xA001

Frequent cases:

Most used CRCs	Representation	NORMAL	REVERSE	RECIPROCAL	Used by
CRC-16-CCITT	x16 + x12 + x5 + 1	0x1021	0x8408	0x0811	Modem X25, Bluetooth, PPP
CRC-16-IBM	x16 + x15 + x2 + 1	0x8005	0xa001	0x4003	USB, MODBUS, altri

Result	Returns the calculated CRC value
Validity	Rule, Task
Note	By repeatedly calling the function with initial value equal to the calculated CRC value, one can perform incremental calculation of the CRC of a buffer of any length.

© 2025 Robox SpA

Syntax	**u16 crc16_le (string strData, i32 nCount [, u16 initVal=0xFFFF] [, u16 polin=0xA001])**
	NOTE: The previous crc16_ero function (Little Endian with unmodifiable initial polynomial 0xA001) is deprecated, but remains for compatibility.

The calculation is done according to the Little Endian algorithm.

Examples

Example of use:

```
crcData = crc16_le(data, numBytes, crcInit, pol)
```

Value crcData	data *(strData)*	numBytes *(nCount)*	crcInit *(initVal)*	pol *(polin)*
38392	"d ve"	4	0xFFFF	0xA001
45560	"d ve"	4	0	0xA001
2126	"d ve"	4	0	0x1021
63824	"13564"	5	0xFFFF	0xA001
63860	"13564"	5	0	0xA001

Function ct_cv()

Function to read the current value of the counter.

NOTE: The similar instruction ct_cv() is used instead to set the current value of the counter.

Syntax	**real ct_cv (counter cnt)**
cnt	Counter name (defined as type of variable)
Result	Returns the value of the counter
Validity	Rule, Task
Note	Useful for creating retentive counters

Examples

Example of use:

```
ct_cv(C1, rr(1000)) ; I write the value contained in the real
register rr(1000) into the current value of counter C1
currVal = ct_cv(C1) ; I read the current value of the counter and
write it into the variable currVal
```

© 2025 Robox SpA

Function ct_pv()

Instruction to read the preset value of the counter.

Note: The similar Instruction ct_pv 675() is used instead to slip the counter preset value.

Syntax	**real ct_pv (counter cnt)**
cnt	Name of counter 892 (defined as type of variable)
Result	Returns the value of the preset
Validity	Rule, Task
Note	

Examples

Example of use:

```
ct_pv 270(C1, rr(1000)) ; I write the value contained in the real
register rr(1000) into the preset of counter C1
presetVal = ct_pv(C1) ; I read the value of the preset and write it
into the presetVal variable
```

Function ct_qd()

Function to read the status value of the counter output (compared with the value 0).

Syntax	**i32 ct_qd (counter cnt)**
cnt	Name of counter 892 (defined as type of variable)
Result	Returns the output state of the counter
Validity	Rule, Task
Note	

Examples

Example of use:

```
out(257) = ct_qd 270(C1) ; out 257 on if current value ct_cv() is <=0
out(257) = ct_qd(C1) and AUTO ; out 257 on if current value ct_cv()
is <=0 and AUTO variable is TRUE
```

© 2025 Robox SpA

Function ct_qu()

Function to read the status value of the counter output (relative to the preset (ct_pv 270 ())).

Syntax	**i32 ct_qu (counter cnt)**
cnt	Name of counter 892 (defined as type of variable)
Result	Returns the output state of the counter
Validity	Rule, Task
Note	

Examples

Example of use:

```
out(257) = ct_qu 271(C1) ; out 257 on if preset value ct_pv() is <=0
out(257) = ct_qu(C1) and AUTO ; out 257 on if preset value ct_pv() is
<=0 and AUTO variable is TRUE
```

Function dec()

Decreases a variable by the specified unit and also returns its new content.

Syntax	**real dec (real var [, real value=1])**
var	Variable to be decremented
value	Decrement value. (optional, default 1)
result	Returns the new content of *var* (result of subtraction)
Validity	Rule, Task
Note	The result can also be omitted

Examples

Example of use:

```
pluto = 150
rr(1) = 5
rr(1) = dec 271(rr(1))
rr(2) = dec(pluto, 100)
; rr(1) will be equal to 4
; rr(2) will be equal to 50

for(i=8, i > 0, dec(i))
; corresponds to for(i=8, i>0, i=i-1)
```

© 2025 Robox SpA

```
appo = dec(pluto, 10)
; corresponds to:
; pluto = pluto - 10
; appo  = pluto
```

Function delta_enc16()

This function is used to perform the calculation of the difference between two integer units, properly handling the "unwrapping" of data whenever the value of an incoming parameter has exceeded the range of the unit used.

Syntax	**real delta_enc16 (real actEnc, real oldEnc)**
actEnc	Contains the first unit (current value). (16-bit int format: range -32768-:-32767)
oldEnc	Contains the unit to be subtracted from the first (previous value). (16-bit int format: range -32768-:-32767)
Result	Returns the difference *actEnc* - *oldEnc*, modulated as if it were a 16-bit integer type variable
Validity	Rule, Task
Note	

A typical example of the function's use is to construct a real quota based on an incremental encoder reading.

In the case of encoders, if *actEnc* is the current encoder reading and *oldEnc* is the reading taken in the previous sampling, the function returns the number of bits to which the encoder has moved in the period between the two readings.

Examples

Example of use:

```
real quotaAsse
real actEnc, oldEnc
; read encoder (from rule)
oldEnc = actEnc
actEnc = inp_w(xxx)    ; vedi file RHW.cfg
quotaAsse = quotaAsse + delta_enc16 272 (actEnc, oldEnc)

; var real 32bitHex I32
argAct = 1000.0 0x000003e8
argOld = 900.0  0x00000384
reso   = 100    0x00000064

argAct = 900.0  0x00000384
argOld = 1000.0 0x000003e8
```

© 2025 Robox SpA

```
reso   = -100   0xffffff9c

argAct = 10.0   0x0000000a
argOld = -10.0  0xfffffff6
reso   = 20     0x00000014

argAct = -10.0  0xfffffff6
argOld = 10.0   0x0000000a
reso   = -20    0xffffFFEC

argAct = -2147483638 0x8000000A
argOld = 2147483638  0x7ffffff6
reso   = 20          0x00000014

argAct = 2147483638  0x7ffffff6
argOld = -2147483638 0x8000000A
reso   = -20         0xFFFFFFEC
```

Function delta_enc32()

This function is used to perform the calculation of the difference between the two arguments, after reducing them to 32-bit entities of type fix.

Syntax	**real delta_enc32 (real actEnc, real oldEnc)**
actEnc	Contains the first unit (current value). (32-bit int format: range -2147483648-:-2147483647)
oldEnc	Contains the unit to subtract from the first (previous value). (32-bit int format: range -2147483648-:-2147483647)
Result	Returns the difference *actEnc* - *oldEnc*, modulated as if it were a 32-bit integer type variable
Validity	Rule, Task
Note	

A typical example of the function's use is to construct a real quota based on an incremental encoder reading.

In the case of encoders, if *actEnc* is the current encoder reading and *oldEnc* is the reading taken in the previous sampling, the function returns the number of bits to which the encoder has moved in the period between the two readings.

Examples

Example of use:

```
real quotaAsse
real actEnc, oldEnc
; read encoder (from rule)
```

© 2025 Robox SpA

```
oldEnc = actEnc
actEnc = inp_w(xxx)      ; see file RHW.cfg
quotaAsse = quotaAsse + delta_enc32 273 (actEnc, oldEnc)

; var real 32bitHex I32
argAct = 1000.0 0x000003e8
argOld = 900.0  0x00000384
reso   = 100    0x00000064

argAct = 900.0  0x00000384
argOld = 1000.0 0x000003e8
reso   = -100   0xffffff9c

argAct = 10.0   0x0000000a
argOld = -10.0  0xfffffff6
reso   = 20     0x00000014

argAct = -10.0  0xfffffff6
argOld = 10.0   0x0000000a
reso   = -20    0xffffFFEC

argAct = -2147483638 0x8000000A
argOld = 2147483638  0x7ffffff6
reso   = 20          0x00000014

argAct = 2147483638  0x7ffffff6
argOld = -2147483638 0x8000000A
reso   = -20         0xFFFFFFEC
```

Function diff()

Checks if a variable or expression has changed its value since the previous analysis.

Syntax	**i32 diff (var)**
var	Parameter to be checked. Must be of type numeric (i8, i16, i32, u8, u16, u32, float, real) or boolean (logical expression)
Result	Returns: Value: <> 0 — Variable changed value from previous analysis Value: 0 — Otherwise
Validity	Rule, Task

Syntax	**i32 diff (var)**
Note	The first time it is executed, the function returns FALSE and stores the current state of the condition. In case diff() is used in a user R3 function, it should be avoided that the parameters of the function itself are involved in the "condition" expression. The result would not be as hoped because of the overlap in memory of the state of multiple variables

Each diff() instruction has its own memory of the "condition" state.

The evaluation of the front is done between the current state of the condition and the memory related to the single function call.

For example, if we have at step 1000 `if(diff(inp(1 )))` and at step 1100 `if(diff(inp(1)))`, though they refer to the same condition, the evaluations are done separately, each according to the stored state, by executing step 1000 and then step 1100.

Examples

Example of use:

```
_if(diff 274(r(1)))
    sr(1) = "È cambiato lo stato"

if(diff(rModoOperativo))
    call aggiornaValori
endif

; equal to:
int appo
appo = rmodoOperativo
loop:
    if(rmodoOperativo <> appo)
        call aggiornaValori
    endif
    appo = rmodoOperativo
    ...
    ...
    ...
goto loop
```

Function dw0()

This function is used to read or write (Instruction) the first dword (starting with the least significant) of the argument.

Reading:

Syntax	**u32 dw0 (i32|u32|real arg)**
arg	Arg of the first dword.

© 2025 Robox SpA

Syntax	**u32 dw0 (i32|u32|real arg)**
	From RTE 33.7.1 it can be any expression
Result	Returns the dword of *arg*
Validity	Rule, Task
Note	Since RTE 33.9.0 accessing a non-existing dword returns an error

Writing (Instruction):

Syntax	**dw0 (i32|u32|real arg) = u32 val**
arg	Arg of the first dword
value	Value to be imposed in *arg*
Validity	Rule, Task
Note	Since RTE 33.9.0 accessing a non-existing dword returns an error

Example

Example of use:

```
int pippo1, pippo2, pippo3, pippo4
int prova[4]
prova[0] = 0x1122
prova[1] = 0x3344
prova[2] = 0x5566
prova[3] = 0x7788
pippo1 = dw0 275 (prova[0]) ; pippo1 = 0x33441122
pippo2 = dw1 277 (prova[0]) ; pippo2 = 0x77885566
pippo3 = dw0(prova[2]) ; pippo3 = 0x77885566
pippo4 = dw1(prova[2]) ; pippo4 = 0x00000000

STRING pippo[26]
STRING pluto[26]
pippo = "abcdefghilmnopqrstuv12345"

dw0 676 (pluto[0]) = dw1(pippo[0])
dw1 677 (pluto[0]) = dw0(pippo[0])
; in pluto I will have "efghabcd"

; with RTE 33.9.0:
i16 pippo = 123
dw1(pippo) = 9  ; error!
```

Function dw1()

This function is used to read or write (instruction) the second dword (starting with the least significant one) of the argument.

Reading:

Syntax	**u32 dw1 (real arg)**
arg	Arg of the second dword. From RTE 33.7.1 it can be any expression
Result	Returns the dword of *arg*
Validity	Rule, Task
Note	Since RTE 33.9.0 accessing a non-existing dword returns an error

Writing (Instruction):

Syntax	**dw1 (real arg) = u32 val**
arg	Arg of the second dword
value	Value to be imposed in *arg*
Validity	Rule, Task
Note	Since RTE 33.9.0 accessing a non-existing dword returns an error

Function dwbe2le()

Converts big endian dword (x32) to little endian dword (x32).

Syntax	**i32|u32 dwbe2le (i32|u32 data)**
data	Data input
Result	Return value
Validity	Rule, Task
Note	

Examples

Example of use:

```
u32 input
u32 result

input = 0x01020304
result = dwbe2le(input)
; result = 0x04030201
```

Function dwbe2lef()

Converts big endian dword (x32) to little endian float (4 bytes).

Syntax	**float dwbe2lef (i32|u32 data)**
data	Data input
Result	Return value
Validity	Rule, Task
Note	

Examples

Example of use:

```
u32 input
float result

input = 0xdb0f4940     ; data memory: 40 49 0f db hex
result = dwbe2lef(input)
; result = 3.1415927 -> data memory: db 0f 49 40 hex
```

Function dwbe2mbe()

Converts big endian dword (x32) to middle big endian dword (x32).

Syntax	**i32|u32 dwbe2mbe (i32|u32 data)**
data	Data input
Result	Return value
Validity	Rule, Task
Note	

Examples

Example of use:

```
u32 input
u32 result

input = 0x01020304
result = dwbe2mbe(input)
; result = 0x03040102
```

© 2025 Robox SpA

Function dwbe2mle()

Converts big endian dword (x32) to middle little endian dword (x32).

Syntax	**i32|u32 dwbe2mle (i32|u32 data)**
data	Data input
Result	Return value
Validity	Rule, Task
Note	

Examples

Example of use:

```
u32 input
u32 result

input = 0x01020304
result = dwbe2mle(input)
; result = 0x02010403
```

Function dwle2be()

Converts little endian dword (x32) to big endian dword (x32).

Syntax	**i32|u32 dwle2be (i32|u32 data)**
data	Data input
Result	Return value
Validity	Rule, Task
Note	

Examples

Example of use:

```
u32 input
u32 result

input = 0x04030201
result = dwle2be(input)
; result = 0x01020304
```

© 2025 Robox SpA

Function dwle2bef()

Converts little endian float (4 bytes) to big endian dword (u32).

Syntax	**u32 dwle2bef (float data)**
data	Data input
Result	Return value
Validity	Rule, Task
Note	

Examples

Example of use:

```
float input
u32 result

input = 3.1415927 ; data memory: db 0f 49 40 hex
result = dwle2bef(input)
; result = 0xdb0f4940 -> data memory: 40 49 0f db hex
```

Function dwle2mbe()

Converts little endian dword (x32) to middle big endian dword (x32).

Syntax	**i32|u32 dwle2mbe (i32|u32 data)**
data	Data input
Result	Return value
Validity	Rule, Task
Note	

Examples

Example of use:

```
u32 input
u32 result

input = 0x04030201
result = dwle2mbe(input)
; result = 0x03040102
```

© 2025 Robox SpA

Function dwle2mbef()

Converts little endian float (4 bytes) to middle big endian dword (u32).

Syntax	**u32 dwle2mbef (float data)**
data	Data input
Result	Return value
Validity	Rule, Task
Note	

Examples

Example of use:

```
float input
u32 result

input = 3.1415927 ; data memory: db 0f 49 40 hex
result = dwle2mbef 281 (input)
; result = 0x4940db0f -> data memory: 0f db 40 49 49 hex
```

Function dwle2mle()

Converts little endian dword (x32) to middle little endian dword (x32).

Syntax	**i32|u32 dwle2mle (i32|u32 data)**
data	Data input
Result	Return value
Validity	Rule, Task
Note	

Examples

Example of use:

```
u32 input
u32 result

input = 0x04030201
result = dwle2mle 281 (input)
; result = 0x02010403
```

© 2025 Robox SpA

Function dwle2mlef()

Converts little endian float (4 bytes) to middle little endian dword (u32).

Syntax	**u32 dwle2mlef (float data)**
data	Data input
Result	Return value
Validity	Rule, Task
Note	

Examples

Example of use:

```
float input
u32 result

input = 3.1415927 ; data memory: db 0f 49 40 hex
result = dwle2mlef 282 (input)
; result = 0x0fdb4049 -> data memory: 49 40 db 0f hex
```

Function dwmbe2be()

Converts middle big endian dword (x32) to big endian dword (x32).

Syntax	**i32|u32 dwmbe2be (i32|u32 data)**
data	Data input
Result	Return value
Validity	Rule, Task
Note	

Examples

Example of use:

```
u32 input
u32 result

input = 0x03040102
result = dwmbe2be 282 (input)
; result = 0x01020304
```

© 2025 Robox SpA

Function dwmbe2le()

Converts middle big endian dword (x32) to little endian dword (x32).

Syntax	**i32|u32 dwmbe2le (i32|u32 data)**
data	Data input
Result	Return value
Validity	Rule, Task
Note	

Examples

Example of use:

```
u32 input
u32 result

input = 0x04030201
result = dwmbe2le(input)
; result = 0x03040102
```

Function dwmbe2lef()

Converts middle big endian dword (u32) to little endian float (4 bytes).

Syntax	**float dwmbe2lef (u32 data)**
data	Data input
Result	Return value
Validity	Rule, Task
Note	

Examples

Example of use:

```
u32 input
float result

input = 0x4940db0f ; data memory: 0f db 40 49 hex
result = dwmbe2lef(input)
; result = 3.1415927 -> data memory: db 0f 49 40 hex
```

© 2025 Robox SpA

Function dwmle2be()

Converts middle little endian dword (x32) to big endian dword (x32).

Syntax	**i32|u32 dwmle2be (i32|u32 data)**
data	Data input
Result	Return value
Validity	Rule, Task
Note	

Examples

Example of use:

```
u32 input
u32 result

input = 0x02010403
result = dwmle2be(input)
; result = 0x01020304
```

Function ecat_dev_on_off()

This function enables/disables a device on the EtherCAT line.

Note: In order to enable/disable the device, the line must be in the INIT state(ecat_set_state(*nEth*, 1)).

Syntax	**i32 ecat_dev_on_off (i32 nEth, i32 cfgAddr , bool sts)**
nEth	Ethernet channel number
cfgAddr	Device address
sts	Desired status: Value: TRUE — Device enabled Value: FALSE — Device not enabled
Result	Returns: Value: 0 — Operation completed Value: -1 — Illegal Ethernet channel number

© 2025 Robox SpA

Syntax	**i32 ecat_dev_on_off (i32 nEth, i32 cfgAddr , bool sts)**
	<table><tr><th>Value</th><th>Description</th></tr><tr><td>-2</td><td>Ethernet channel is not EtherCAT</td></tr><tr><td>-3</td><td>EtherCAT not in INIT state</td></tr><tr><td>-4</td><td>No device in the selected location</td></tr><tr><td>-5</td><td>EtherCAT line not properly configured</td></tr><tr><td>-6</td><td>Internal sw error. (contact Robox)</td></tr></table>
Validity	Task
Note	

Examples

Example of use:

```
;_____________________________________________________________________
_____________________
FUNCTION int Ecat_Change_Config(int nEth, int addr, bool sts, bool
restartEcat, int@ phase)

; it performs the following operations :
; 1 set the ethercat in INIT state
; 2 change the device status
; 3 wait for the restart Ecat flag
; 4 set the ethercat in OPERATIONAL state
; 5 wait for OPERATIONAL state is reached, if not an error is set
; 6 when OPERATIONAL state is reached, the alarm due to this
operation are reset
;
; parameters
; nEth  number of ethernet (1 -:-...)
; addr  device config_addr
; sts   1 --> device present  0 --> device absent
; restartEcat  flag to command the restart of the ethercat line (to
be set after that the
;               device has been physically removed or restored to
ethercat line)
; phase  working fase to be set to 0 to restart the opreation
; return
;   0   operation in execution
```

```
;   1   operation to change the status done, waiting to 'restartEcat'
;   2   operation to restart ethercat line  is done
;  -1   no such device
;  -2   illegal phase
;  -3   unable to  restart ecat , check device
    select(phase)
        default
            phase = -2
        case -2
            return -2        ; illegal phase

        case -3
            return -3        ; unable to restart ecat , check device

        case -1
            return -1        ; error .. no such device

        case 0
            ; set ethercat line in init mode
            _if(nEth = 0)        ; 0 means alla ethercat lines
                nEth = -1        ; not acceptable here
            esitoECC = ecat_set_state(nEth, 1)
            if (esitoECC = 0)
                phase = 1
            endif
            return 0
        case 1  ; wait the init state
            if(ecat_get_state(nEth) <> 1)
                return 0
            endif
            ; now the modification are allowed
            esitoECC = ecat_dev_on_off(nEth, addr, sts)
; $lin_step esitoecc = ECATSlaveOnOff(neth, addr, sts);
            if(esitoECC < 0)
                phase = -1
                return -1
            endif

            phase = 2

        case 2
            if(not restartEcat)
                return 1
            endif
            ; set ethercat line in operational mode
            esitoECC = ecat_set_state(nEth, 8)
            if(esitoECC = 0)
                phase = 3
                ecatTimeOut = tfb + 10
            endif
            return 0
```

```
        case 3
            ecatState = ecat_get_state(nEth)
            if(ecatState = 1)
                if(tfb < ecatTimeOut)
                    return 0
                endif
                ; too much time to change from init state to
preopreational....
                ; something wrong
                phase = -3
                return -3
            endif
            ; wait the operational state
            if(ecatState <> 8)
                return 0
            endif
            ; ok the ethercat line is in operational....
            ; clean some alarms due to the operation
            alarm_reset(18, 0)
            alarm_reset(82)
            alarm_reset(83)
            phase = 4

        case 4
            return 2
    end_select
    return -2
END_FUN
```

Function ecat_get_state()

Function that returns the NMT operating mode of the EtherCAT line indicated.

Syntax	**i32 ecat_get_state (i32 nEth)**
nEth	EtherCAT line number
Result	Returns:

Value	Description
1	INIT
2	PREOPERATIONAL
4	SAFEOPERATIONAL
8	OPERATIONAL
-1	Illegal EtherCAT Line Index

© 2025 Robox SpA

Syntax	**i32 ecat_get_state (i32 nEth)**
	Value: -2 Description: Ethernet port not configured EtherCAT
Validity	Rule, Task
Note	

Examples

Example of use:

```
....
; Restart the Ethercat line  on ETH 2
; force the line to INIT STATE
result = ecat_set_state(2, 1)       ; set line 2 in init
if (result < 0)
    alarm_set(UNABLE_TO_SET_ECAT, 0, 1)
    return
endif
mt_wait(ecat_get_state(2) = 1 )     ; wait for line in init state
; ok now restart
result = ecat_set_state(2, 8)       ; set line 2 in operational
if (result < 0)
    alarm_set(UNABLE_TO_SET_ECAT, 0, 2)
    return
endif
mt_wait(ecat_get_state(2) = 8 )     ; wait for line in operational
state
...
```

NOTE: This example can only be used in the task, as the mt_wait() instruction is used.

Function ecat_probe()

Function to read the information of the device connected in EtherCAT line.

Instruction takes a long time. It must be called continuously until it is finished.

Syntax	**i32 ecat_probe (i32 nEth, stru_ecat_probe stru [, string devName=""])**
nEth	EtherCAT channel number
stru	Structure of type stru_ecat_probe that contains the information
devName	String where the device name is saved. (optional, default "")

Syntax	**i32 ecat_probe (i32 nEth, stru_ecat_probe stru [, string devName=""])**
Result	Returns:

Value	Description
1	Operation completed
0	Running operation
-1	Illegal EtherCAT Line Index
-2	Ethernet channel is not EtherCAT
-3	Illegal topic
-4	No device in the selected location
-5	EtherCAT line not configured correctly
-6	EtherCAT cable disconnected
-7	Internal sw error. (contact Robox)

Validity	Task
System Requirements.	RC3E 33.7.1-beta0
Note	

Examples

Example of use:

```
; storage for all the ethercat device in a line (up to
N_MAX_ECAT_DEV)
$LIT N_MAX_ECAT_DEV     100
stru_ecat_probe 814 ecatDev[1+N_MAX_ECAT_DEV]       ; structure to
store information minimum index is 1
string devName[1+N_MAX_ECAT_DEV][64]            ; device name
int nDev                                        ; number of devices
found
int esitoEP                                     ; result of
ECAT_PROBE
string appoName[64]                             ; temporary device
name
...
...
;_____________________________________________
```

```
FUNCTION void hndEcatProbe(int nEth)
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
; function to probe all the devices of an ethercat channel
    int howMany = 0
    nDev = 1
    do
        ecatDev[ndev].phase =  0
        ecatDev[ndev].n_Pos  = nDev
        esitoEp = 0
        while(esitoEp = 0)
            esitoEp = ecat_probe 288 (nEth, ecatDev[nDev], apponame)
    ;$lin_step esitoep = ECATProbe(neth, &ecatdev[ndev], apponame,
sizeof(apponame)  );
        end_while
        if(esitoEp = 1)
            howMany = ecatDev[ndev].how_many
            devName[nDev] = apponame
            nDev = nDev+1
        endif
    end_do_while((esitoEp =1) and (nDev <= N_MAX_ECAT_DEV))
    if(nDev = 1)
        nDev = 0
    else
        if(nDev > howMany)
            nDev = howMany
        endif
    endif
END_FUN
```

Function ecat_set_state()

Function that imposes the NMT operating mode of the indicated EtherCAT line.

<table>
<tr><th>Syntax</th><th>i32 ecat_set_state (i32 nEth, i32 desState [, bool doBopex=FALSE])</th></tr>
<tr><td>nEth</td><td>EtherCAT line number.
If 0 means all EtherCAT lines present.</td></tr>
<tr><td>desState</td><td>State of the network you wish to impose:
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>1</td><td>INIT</td></tr>
<tr><td>2</td><td>PREOPERATIONAL</td></tr>
<tr><td>4</td><td>SAFEOPERATIONAL</td></tr>
<tr><td>8</td><td>OPERATIONAL</td></tr>
</table></td></tr>
<tr><td>doBopex</td><td>Se uguale a TRUE, attende di essere in SAFEOPERATIONAL prima di eseguire il file</td></tr>
</table>

© 2025 Robox SpA

Syntax	**i32 ecat_set_state (i32 nEth, i32 desState [, bool doBopex=FALSE])**
	comandi bopex.stp. Altrimenti non verrà eseguito il file. (opzionale, default FALSE)
Result	Returns:

Value	Description
2	OK: SAFEOPERATIONAL reached, bopex.stp executed
1	OK: SAFEOPERATIONAL achieved, bopex.stp not present
0	OK: desired state imposed
-1	EtherCAT line index illegal
-2	Ethernet port not configured EtherCAT
-3	Illegal status code
-4	Network not started due to configuration error. (use mreport -a)
-5	Function used not in a timesharing task with *doBopex* TRUE flag.
-6	Timeout for waiting for SAFEOPERATIONAL status to then execute bopex.stp: operation not executed

Validity	Rule, Task
Note	

Examples

Example of use:

```
....
; Restart the Ethercat line on ETH 2
```

© 2025 Robox SpA

```
; force the line to INIT STATE
result = ecat_set_state(2, 1)       ; set line 2 in init
if (result < 0)
    alarm_set(UNABLE_TO_SET_ECAT, 0, 1)
    return
endif
mt_wait(ecat_get_state(2) = 1 )     ; wait for line in init state
; ok now restart
result = ecat_set_state(2, 8)       ; set line 2 in operational
if (result < 0)
    alarm_set(UNABLE_TO_SET_ECAT, 0, 2)
    return
endif
mt_wait(ecat_get_state(2) = 8 )     ; wait for line in operational
state
...
```

NOTE: This example can only be used in the task, as the mt_wait() instruction is used.

Function excl_lam()

Function to obtain the mask of the excluded/included logical axes of the PowerSet.

Syntax	**i32 psName.excl_lam**
psName	PowerSet name
Result	Returns the mask of the excluded/included logical axes of the PowerSet. (1 excluded, 0 included)
Validity	Rule, Task
Note	

Examples

Example of use:

```
; in cfgdefs.i3
...
; {7:0} Power set variables
...
power_set sys
power_set ps_qua          ; powerset definition
...

; in task o rule
int maskAx                ; axes mask excluded/reincluded
...
...
```

© 2025 Robox SpA

```
maskAx = ps_qua.excl_lam 292
...
```

Function excl_pam()

Function to obtain the mask of the excluded/included physical axes of the PowerSet.

Syntax	**i32 psName.excl_pam**
psName	PowerSet name
Result	Returns the mask of the excluded/included physical axes of the PowerSet. (1 excluded, 0 included)
Validity	Rule, Task.
Note	

Examples

Example of use:

```
; in cfgdefs.i3
...
; {7:0} Power set variables
...
power_set sys
power_set ps_qua        ; powerset definition
...

; in task o rule
i32 maskAx              ; axes mask excluded/reincluded
...
...
maskAx = ps_qua.excl_pam 293
...
```

Function f_avail()

This function is used to display the flash card memory available on the various logic devices.

Syntax	**real f_avail (string path)**
path	String containing the name of a directory
Result	Returns a value containing the number of free bytes on the given *path*
Validity	Task

© 2025 Robox SpA

Syntax	**real f_avail (string path)**
System Requirements	**WARNING**: This function is not valid for RDE versions >= 33.
Note	If functions regarding compact flash management are used, it is recommended that bit 14 of the sys_cfg variable (0x00004000) be imposed to 1, to enable the use of safety routines for flash write/read

Examples

Example of use:

```
esito = f_avail(/FA/)
```

Function f_close()

This function allows you to close a file on flash eprom previously opened by the instruction f_open().

Syntax	**i32 f_close (stru_filehnd stru)**
stru	Variable of type stru_filehnd that contains the handle obtained via the f_open() function
Result	Returns: Value — Description 0 — Positive result of the operation EOF (code 0x1a) — Negative result of the operation
Validity	Task
Note	If functions regarding compact flash management are used, it is recommended that bit 14 of the sys_cfg variable (0x00004000) be imposed to 1 to enable the use of safety routines for flash write/read

Examples

Example of use:

```
int result_close
int result_open
string result
stru_filehnd cf
```

```
result_open = f_open("/fa/ipaddr2.def", "wb", 0, cf)
...
result_close = f_close(cf)
if(result_close <> 0)
    result = "function f_close: negative result"
endif
```

Function f_copy()

This function is used to copy the contents of a file, present on the CF of the connected motion control, to another specified file.

Syntax	**i32 f_copy (string srcPathName, string destPathName)**
srcPathName	Current name of the file including path
destPathName	Desired name of the file including path. Destination path can also be different
Result	Returns: Value / Description 0 / Positive result of the operation -21 / The target file already exists or is a directory -17 / The destination file already exists -2 / The source file does not exist -nn / ERROR error code. (see error codes)
Validity	Task
Note	This function blocks the evolution of the task until the task is finished. Keep in mind that copying a 120K file takes about 1.35 sec on uRmc2 and about 0.8 sec on uRmc3. If functions regarding compact flash management are used, it is recommended that bit 14 of the sys_cfg variable (0x00004000) be imposed to 1 to enable the use of the safety routines for flash write/read

Examples

Example of use:

```
int result_copy
string result

result_copy = f_copy("/fa/test.def", "/fa/test1.def")
if(result_copy = 0)
   result = "function f_copy: positive result"
endif
```

Function f_del()

This function is used for deleting one or more files in a partition.

Syntax	**i32 f_del (string fileName)**
fileName	Full path to the directory/file you want to delete. /fx/filename where: • x is the logical device identifier character (@, A, B, C, D, E) • filename is the name of the file with wildcards (* and ?)
Result	Returns: Value: > 0 — Number of files that have been deleted Value: < 0 — Error code (see error codes)
Validity	Task
Note	If functions regarding compact flash handling are used, it is recommended that bit 14 of the sys_cfg variable (0x00004000) be imposed to 1, to enable the use of safety routines for flash write/read

Examples

Example of use:

```
int result_del
string result

result_del = f_del("/fa/test.def")
if(result_del<=0)
   result = "function f_del: error"
endif
```

Function f_dir()

This function allows you to obtain certain information about the files present in flash.

Syntax	**i32 f_dir (string path, i32 ndef=0 [, stru_fileinfo stru[]=NULL])**
path	Path to the directory containing the desired file (or more than one) or to the file itself. If you want to refer to a specific file you must put the full path (e.g., "/fa/override.cfg"). If you want to refer to one (not directly) or more files you have to use the * character, where * replaces any character or string (e.g., "/fa/*.cfg" I request information about all files, regardless of name, with the cfg extension contained in the fa folder). The folder identifier characters can be @, A, B, C, D, E, ... (e.g. /f@/, /fb/, ...). **NOTE1**: If I enter the full name of a file in the folder, I get information about the size of the file itself (e.g., "/fa/stat.txt" I will get the size of the file stat.txt). **NOTE2**: If you are using an old flash-card with a Pentium CPU, then the *folder* field will only be /fx/ and not /fx/*.*.
nDef	Parameter containing the number of desired information (5 = 5 files) (optional, default 0)
stru	Array of stru_fileinfo 816 that will contain information about the files found. (optional, default NULL--> desired information will not be saved) Name and size of files found in folder defined by *path* will be saved. Must be at least *nDef* elements
Result	Returns the number of files present in the given partition
Validity	Task
Note	If functions regarding compact flash management are used, it is recommended that bit 14 of the sys_cfg variable (0x00004000) be imposed to 1, to enable the use of the safety routines for writing/reading flash

Examples

Example of use:

```
; Suppose we have the folder /fb/ with two files in it: test1.def and
test2.def

int result
stru_fileinfo 816 infoFile[2]

result = f_dir 297 ("/FB/*.*")
; result = 2

result = f_dir("/FB/test1.def")
; result = 1

result = f_dir("/FB/*.*", 0, infofile)
; result = 2
; infofile[0].name = ""
; infofile[0].len = 0
; infofile[1].name = ""
; infofile[1].len = 0

result = f_dir("/FB/*.*", 2, infofile)
; result = 2
; infofile[0].name = "test1.def"
; infofile[0].len = 161
; infofile[1].name = "test2.def"
; infofile[1].len = 30

result = f_dir("/FB/test1.def", 2, infofile)
; result = 1
; infofile[0].name = "test1.def"
; infofile[0].len = 161
; infofile[1].name = ""
; infofile[1].len = 0
```

Function f_format()

WARNING: This function is not valid for RDE versions >= 33.

This function is used to perform formatting of a logical device.

Syntax	**i32 f_format (string path [, string result=NULL] [, real timeOut=0])**
path	Parameter containing the name of the path. It can take the following characters: • /FB/ • /FC/ Use fmem directive from shell to view the partitions present on the flash

Syntax	**i32 f_format (string path [, string result=NULL] [, real timeOut=0])**
result	Parameter containing the result of the operation. (optional, default NULL) Can be: Value / Description "Ok" / Operation terminated successfully "timeout system not available" / Manager busy or timeout expired "illegal or not allowed device" / wrong, nonexistent or not allowed *path* (F@, FA)
timeOut	Parameter containing the maximum waiting time for instruction execution. (optional, default 0)
Result	Returns: Value / Description <> 0 / Operation performed successfully 0 / Operation failed Use the optional *result* and *timeOut* parameters for more diagnostics
Validity	Task
System Requirements	
Note	All contents of the device in question are erased. RTE prevents the formatting of F@ and FA. If functions regarding compact flash management are used, it is recommended that bit 14 of the sys_cfg variable (0x00004000) be imposed to 1, to enable the use of safety routines for writing/reading flash

Diagnostics

- Generate alarm 9900 illegal arg A:1 T:x ST:xxx if illegal *path* parameter (where T = number of tasks and ST number of steps).

Examples

Example of use:

© 2025 Robox SpA

```
string esitoOper
reso = f_format(298)("/fb/")
reso = f_format("/fb/", esitoOper)
reso = f_format("/fb/", esitoOper , 5)
f_format("/FC/")
```

Function f_gets()

This function is used to read a block of data from a file that has been opened in text read mode.

Syntax	**i32 f_gets (string buff, i32 maxLen, stru_filehnd stru)**
buff	In *buff* the string read from the file is copied. The read ends when the line terminator character LF (Line Feed, code 0X0A) is encountered. The LF character is put as the last character in the string
maxLen	Specifies the maximum size of each string read from the file (considering the terminator character). **Note**: The buffer must be at least one character larger than *maxLen*.
stru	Variable of type stru_filehnd(815) that contains the handle obtained via the f_open(302)() function.
Result	Returns: Value: 0 — Failed operation Value: 1 — Operation performed successfully
Validity	Task
Note	If there is a string longer than *maxLen* in the file, it will be truncated to the *maxLen* length. The truncated portion will be read the next time f_gets(300)() is called. If functions regarding compact flash management are used, it is recommended that bit 14 of the sys_cfg variable (0x00004000) be imposed to 1, to enable the use of safety routines for writing/reading flash

Examples

Example of use:

```
; Suppose we have in /fa/ the file autoexec.stp, consisting of these
two lines of text:
; sr(1) = "test.def"
; sr(2) = "rt"

int result_gets
int result_open
string gets_buffer
stru_filehnd 815 cf

result_open = f_open 302("/fa/autoexec.stp", "rt", 0, cf)
result_gets = f_gets 300(gets_buffer, sizeof(gets_buffer)-1, cf)
; gets_buffer = sr(1) \t= "test.def"\n
```

Function f_mkdir()

Function to create a new directory in flash.

All directories used in the chosen path, except the last one, must exist in flash.

Syntax	**i32 f_mkdir (string dirName)**
dirName	Name of the new directory. Can contain letters, spaces, and special characters. It is not case sensitive
Result	Returns:

Value	Description
0	Operation performed successfully
-1	Operation failed

Validity	Task
Note	

Examples

Example of use:

```
; making the directory structure:
; machine3
; ... par
; ... ... ax1
; ... ... ax2
; ... ... ax3
; ... ric
; ... doc
esitoF = f_mkdir 301("/machine3")
```

© 2025 Robox SpA

```
if(esitoF = 0)
    esitoF = f_mkdir("/machine3/par")
    if(esitoF = 0)
        esitoF = f_mkdir("/machine3/par/ax1")
        if(esitoF <> 0)
            ; set diagnostic "/machine3/par/ax1"
            ....
        endif
        esitoF = f_mkdir("/machine3/par/ax2")
        if(esitoF <> 0)
            ; set diagnostic on "/machine3/par/ax2"
            ....
        endif
        esitoF = f_mkdir("/machine3/par/ax2")
        if(esitoF <> 0)
            ; set diagnostic on "/machine3/par/ax3"
            ....
        endif
    else
        ; set diagnostic on "/machine3/par"
        ....
    endif
    esitoF = f_mkdir("/machine3/ric")
    if(esitoF <> 0)
        ; set diagnostic "/machine3/ric"
            ....
    endif
    esitoF = f_mkdir("/machine3/doc")
    if(esitof <> 0)
        ; set diagnostic "/machine3/doc"
            ....
    endif
else
        ; set diagnostic on "/machine3"
        ....
endif
```

Function f_open()

This function allows you to open a file on CF for reading or writing and obtain a handle, which is to be used as a parameter for subsequent operations on the same file(f_puts 304(), f_close 294(), ..).

Syntax	**i32 f_open (string fileName, string mode, i32 maxChar, stru_filehnd stru)**
fileName	This is the name (complete with extension) of the file you intend to open. **NOTE**: The name must be preceded by the indication of the logical device you intend to work on, which can be FA, FB, etc. It can be a

Syntax	**i32 f_open (string fileName, string mode, i32 maxChar, stru_filehnd stru)**
	local or global string variable, or a constant string.
mode	This is a parameter that specifies how the file is opened and can take one of the following values:

Value	Meaning
"rb"	Open in read, binary mode
"rt"	Read opening, text mode
"wb"	Writing opening, binary mode
"wt"	Opening in write, text mode
"ab"	Opening in append, binary mode
"at"	Opening in append, text mode

maxChar	**WARNING**: From RTE >= 33 must be imposed equal to zero. This is a parameter that specifies the maximum number of characters that made the file. This value must be given when opening a file for writing, while it must be set to 0 when the file is opened for reading. This parameter is used by the operating system to reserve in flash a fixed portion of memory in which the file will later be written
stru	Variable of type stru_filehnd 815 where the handle characterizing the opened file will be stored and remain associated with it during all subsequent operations, until the file is closed by the instruction f_close 294 (). This variable is to be used as a parameter in all other file handling functions
Result	Returns:

Value	Description
0	Negative result of the operation

Syntax	**i32 f_open (string fileName, string mode, i32 maxChar, stru_filehnd stru)**
	Value: <> 0 — Description: Positive result of the operation
Validity	Task
Note	If functions regarding compact flash management are used, it is recommended to impose bit 14 of the sys_cfg variable (0x00004000) to 1, to enable the use of safety routines for flash writing/reading

Examples

Example of use:

```
int result_open
string result
stru_filehnd 815 cf

result_open = f_open 302("/fa/autoexec.stp", "rb", 0, cf)
if(result_open = 0)
    result = "Not existing file"
endif
```

Function f_puts()

This function is used to write a string to a file that has been opened in write-type text mode.

Syntax	**i32 f_puts (string buff, stru_filehnd stru)**
buff	String to be placed in the file
stru	Variable of type stru_filehnd 815 that contains the handle obtained via the f_open 302() function
Result	Returns:

Value	Description
0	File does not exist
EOF (code 0x1a)	Negative result
> 0	Positive result. The value returned is that of the last byte entered

© 2025 Robox SpA

Syntax	**i32 f_puts (string buff, stru_filehnd stru)**
Validity	Task
Note	If I use a non-empty file, the contents of the file prior to using the function are erased. If using functions regarding compact flash management, it is recommended that bit 14 of the sys_cfg variable (0x00004000) be imposed to 1, to enable the use of safety routines for writing/reading flash

Examples

Example of use:

```
int result_open
int result_puts
string result
stru_filehnd 815 cf

result_open = f_open 302("/fa/test.stp", "wt", 0, cf)
result_puts = f_puts 304("blablabla", cf)
if(result_puts > 0)
   result = "function f_puts: positive result"
endif
```

Function f_read()

This function is used to read a block of data from a file that has been opened in binary read mode.

Syntax	**i32 f_read (buff, i32 blockLen, i32 nBlock, stru_filehnd stru)**
buff	Local or global variable that can be of any type. In buffer the block of data read sequentially from the file is downloaded
blockLen	Size in bytes of each block to be written
nBlock	Number of blocks, of size *blockLen*, to be written
stru	Variable of type stru_filehnd 815 that contains the handle obtained via the function f_open 302()
Result	Returns: Value: 0 — Failed operation

Syntax	**i32 f_read (buff, i32 blockLen, i32 nBlock, stru_filehnd stru)**
	Value: > 0 — Description: The operation was successful. Returns the number of elements actually read. This number usually coincides with the number of blocks, unless the end of the file is encountered before all specified elements have been read
Validity	Task
Note	If you are using functions regarding compact flash management, it is recommended that bit 14 of the sys_cfg variable (0x00004000) be imposed to 1, to enable the use of safety routines for writing/reading flash

Examples

Example of use:

```
int result_open
int result_read
string read_buffer
string result
stru_filehnd 815 cf

result_open = f_open 302("/fa/test.stp", "rb", 0, cf)
result_read = f_read 305(read_buffer, 16, 1, cf)
if(result_read = 0)
    result = "function f_read: negative result"
endif
```

Function f_read_table()

Note: Deprecated. The f_read_table_s 308 function (from RTE 33.21) is recommended.

Reads, from a file present in flash, an array of data that is stored in consecutive real registers or in a specified real array.

Syntax	**i32 f_read_table (string fileName, i32 nEntry, i32 nData, buff)**
fileName	Name of the file to read data from. It must also contain the partition of the flash

© 2025 Robox SpA

Syntax	**i32 f_read_table (string fileName, i32 nEntry, i32 nData, buff)**
nEntry	Number of table entries
nData	Number of data per entry (max 16), that is, the number of data for each row in the file
buff	It must be a variable of one of the following types: • array of variables of type stru_cam 804 • real register rr or nvrr, from which the cam data read from the file must be stored • local array of real variables in which the cam data read from the file is to be stored • array of structures containing only real variables
Result	Returns:

Value	Description
> 0	One or more elements are missing. Returns the number of the row that is missing at least one element
0	Read OK
-1	Read OK. More data in the file
-2	Read OK. Less data in file than expected. Incorrect table
-10	File does not exist
-11	Number of data per line illegal
-12	Number of illegal entries
-14	Empty file

Validity	Task
Note	If functions regarding compact flash management are used, it is recommended that bit 14 of the sys_cfg variable (0x00004000) be imposed to 1, to enable the use of safety routines for flash write/read

The data obtained will then be used in a rule to execute the mv_table 455() and mv_cam 425() functions.

The data contained within the file must respect the following rules:

- maximum line size: 128 characters
- maximum single item size: 18

Examples

Example of use:

```
public stru_cam 804 cerchio[201]  ; I define cerchio as a variable of
type stru_cam (201 elements)
; is defined in a task as public and as extern in the rules
reso = f_read_table 306("FA:cerchio.tab", 201, 2, cerchio)
;where the file loaded on the FA partition of the flash-card consists
of 201 rows of data pairs (2 columns)
; cerchio.tab file
-1 0
-0.99 0.14106736
-0.9800 0.198997487


int reso
string nomefile
reso = f_read_table(nomefile, 201, 2, rr(100))
; rr(100) 1 row 1 column
; rr(101) 1 row 2 column
; rr(102) 2 row 1 column
; rr(103) 2 row 2 column
```

Function f_read_table_s()

Reads from a file present in flash a set of data, which is stored in consecutive real registers or in a specified real array.

It is like f_read_table 306(), but is checked for overflow from the specified buffer *buff*.

An additional difference is that it is not possible to start at any index of an array of reals or structures, but only at the beginning.

The function should be used only by passing correct parameters. It is not recommended to use it normally.

NOTE: If less data is read, the function fills the remainder with the last read.

Syntax	**i32 f_read_table_s (string fileName, i32 nEntry, i32 nData, * buff [, i32 nRows=NULL])**
fileName	Name of the file to read data from. It must also contain the partition of the flash
nEntry	It is the number of entries in the table
nData	Number of data per entry (max 128), i.e., number of data for each line in the file. The line however can be a maximum of 256 characters long

© 2025 Robox SpA

Syntax	**i32 f_read_table_s (string fileName, i32 nEntry, i32 nData, * buff [, i32 nRows=NULL])**
buff	Must be a variable of one of the following types: • array of variables of type stru_cam 804 • real register rr or nvrr, from which the cam data read from the file must be stored • local array of real variables in which the cam data read from the file is to be stored • array of structures that contain only real variables
nRows	Returns the number of valid lines read. (optional, default NULL) (from RTE 34.24.8)
Result	Returns:

Value	Description
> 0	One or more elements are missing. Returns the number of the row that is missing at least one element
0	Read OK
-1	Read OK. More data in the file
-2	Read OK. Less data in file than expected. Incorrect table
-10	File does not exist
-11	Number of data per line illegal
-12	Number of illegal entries
-14	Empty file
-15	Used *nRows* argument with RTE < 34.24.8 (alarm 9900 is generated)

Validity	Task
Note	If functions regarding compact flash management are used, it is recommended that bit 14 of the sys_cfg variable (0x00004000) be imposed to 1, to enable the use of safety routines for flash writing/reading

© 2025 Robox SpA

The data obtained will then be used in the rules for executing the mv_table455() and mv_cam425() functions.

The data contained within the file must respect the following:

- maximum line size: 128 characters
- maximum single item size: 18

Examples

Example of use:

```
public stru_cam804 cerchio[201]  ; I define cerchio as a variable of
type stru_cam (201 elements)
; it is defined in a task as public and as external in the rules
reso = f_read_table_s308("/FA/cerchio.tab", 201, 2, cerchio)
; where the file loaded on the FA partition of the flash-card
consists of 201 rows of data pairs (2 columns)
; file cerchio.tab
-1 0
-0.99 0.14106736
-0.9800 0.198997487

; I take the data from the file residing in /FA/
"file_read_table.txt," consisting of these lines:
; 1
; 2
; 3
; 4
; 5
; 6
; 7
; 8
; 9
; 0
result_table = f_read_table_s("/FA/file_read_table.txt", 10, 1,
rr(100))
; rr(100)=1, rr(101)=2, ... rr(109)=0
```

Function f_rename()

This function is used to rename a file present on the CF of the motion control.

Syntax	**i32 f_rename (string oldPathName, string newPathName)**
oldPathName	Current name of the file including folder
newPathName	Desired name of the file including folder. Destination folder must be the same
Result	Returns:

Syntax	**i32 f_rename (string oldPathName, string newPathName)**
	Value / Description 0 / Positive result of the operation -1 / Negative result
Validity	Task
Note	If functions regarding compact flash management are used, it is recommended that bit 14 of the sys_cfg variable (0x00004000) be imposed to 1, to enable the use of safety routines for flash write/read

Examples

Example of use:

```
int result_rename
string result

result_rename = f_rename(310)("/fa/old_file.txt", "/fa/new_file.txt")
if(result_rename = -1)
   result = "funzione f_rename: esito negativo, vedere report"
else
   result = "funzione f_rename: esito positivo"
endif
```

Function f_seek()

This function is used to move the read/write pointer within a file.

Syntax	**i32 f_seek (i32 offset, i32 mode, stru_filehnd stru)**
offset	Defines the position of the pointer within the file. The value provided by offset is interpreted by the operating system appropriately, depending on the specified *mode* parameter
mode	Is a constant that specifies how the *offset* parameter value is interpreted. Allowed values are as follows: Value / Meaning 0 / Absolute position from the beginning of the file (if *offset=1*, the

Syntax	**i32 f_seek (i32 offset, i32 mode, stru_filehnd stru)**
	<table><tr><th>Value</th><th>Meaning</th></tr><tr><td></td><td>first character of the file is pointed to)</td></tr><tr><td>1</td><td>Relative position from the current pointer position</td></tr><tr><td>2</td><td>Absolute position counting backward from the end of the file (if *offset=1* the last character of the file is pointed to)</td></tr></table>
stru	Variable of type stru_filehnd 815 that contains the handle obtained via the f_open 302() function
Result	Returns 1 if the operation resulted in success
Validity	Task
Note	If functions regarding compact flash management are used, it is recommended that bit 14 of the sys_cfg variable (0x00004000) be imposed to 1 to enable the use of safety routines for flash write/read

Examples

Example of use:

```
int result_open
int result_seek
stru_filehnd 815 cf

result_open = f_open 302("/fa/autoexec.stp", "wb", 0, cf)
result_seek = f_seek 311(1, 2, cf)
; the pointer is on the last character of the autoexec.stp file
```

Function f_write()

This function is used to write a block of data to a file that has been opened in binary write mode.

Syntax	**i32 f_write (buff, i32 blockLen, i32 nBlock, stru_filehnd stru)**
buff	Is a local or global variable. It can be of type string or any other type as long as array. In

Syntax	**i32 f_write (buff, i32 blockLen, i32 nBlock, stru_filehnd stru)**
	buff is contained the block of data to be written sequentially in the file
blockLen	Size in bytes of each block to be written
nBlock	Number of blocks of size *blockLen* to be written
stru	Variable of type stru_filehnd 815 that contains the handle obtained via the f_open 302() function
Result	Returns:

Value	Description
0	Negative result of the operation
<> 0	Positive result

Validity	Task
Note	If functions regarding compact flash management are used, it is recommended that bit 14 of the sys_cfg variable (0x00004000) be imposed to 1, to enable the use of safety routines for flash write/read

NOTE: If I use a non-empty file, the contents of the file prior to using the function are deleted.

Examples

Example of use:

```
; I edit the test.txt file in the /fa/ folder, writing 12345678 into
it.
int result_open
int result_close
int result_write
stru_filehnd 815 cf

result_open = f_open 302("/fa/test.txt", "wb", 0, cf)
result_write = f_write 312("12345678", 8, 1, cf)
result_close = f_close 294(cf)
```

© 2025 Robox SpA

Function fall()

Function to capture the falling edge of a condition.

Syntax	**bool fall (bool cond)**
cond	Condition whose falling edge is to be captured
Result	Returns TRUE only when the condition is FALSE and was TRUE at the previous execution
Validity	Rule, Task
Note	The first time it is executed, the function returns FALSE and stores the current state of the condition. In case fall 314() is used in a user R3 function, it should be avoided that the parameters of the function itself are involved in the condition expression. The result would not be as hoped because of the overlap in memory of the state of multiple variables

Each fall() function has its own memory of the condition state.

The evaluation of the front end is done between the current state of the condition and the memory related to the individual function call (e.g., if we have at step 1000 `if(fall(inp(1 )))` and at step 1100 `if(fall(inp(1)))`, although they refer to the same condition, the evaluation is done separately, each according to the state stored by executing step 1000 and then step 1100).

Examples

Example of use:

```
if(fall(inp(449)))
    sr(1) = "falling edge"
endif

if(fall(r(1)=3))
    sr(1) = "the register r(1) has no more value 3"
endif
```

Function fall_filt()

Function to capture the falling edge of a condition by setting a filter in space or time.

Functioning

Upon detection of the falling edge of the condition, the value of the corresponding filter variable is stored.

As long as the difference between the current value of the filter variable and the stored value is less than the specified threshold, the function returns 0; when the difference

© 2025 Robox SpA

becomes greater than the specified threshold, the function returns the stored value on the falling edge. When the condition is TRUE, the function returns 0.

Syntax	**real fall_filt (bool cond, real varFilt, real thrFilt)**
cond	Condition to be examined for the front search
varFilt	Evolving unit (typically time(tfb) or space ip(n))
thrFilt	Quantity of *varFilt* in which the false condition is to be stably seen
Result	Returns:

Value	Condition
0	*cond* = TRUE
	cond = FALSE but the threshold *thrFilt* was not exceeded
<> 0 (the value of *varFilt* recorded on the falling edge of *cond* is returned)	*cond* = FALSE and you have exceeded *thrFilt* (value of reference variable stored on transition)
	cond = FALSE and delta filter is satisfied (value of quota stored on falling edge)

Validity	Rule, Task
Note	From the first time the function is executed until the condition remains low and the threshold is exceeded the value returned is 0. Past that threshold the value returned is the first supplied value of *varFilt*. In case fall_filt() 314 is used in a user R3 function, it should be avoided that the parameters of the function itself are involved in the 'condition' expression. The result would not be as hoped because of the overlap in memory of the state of several variables

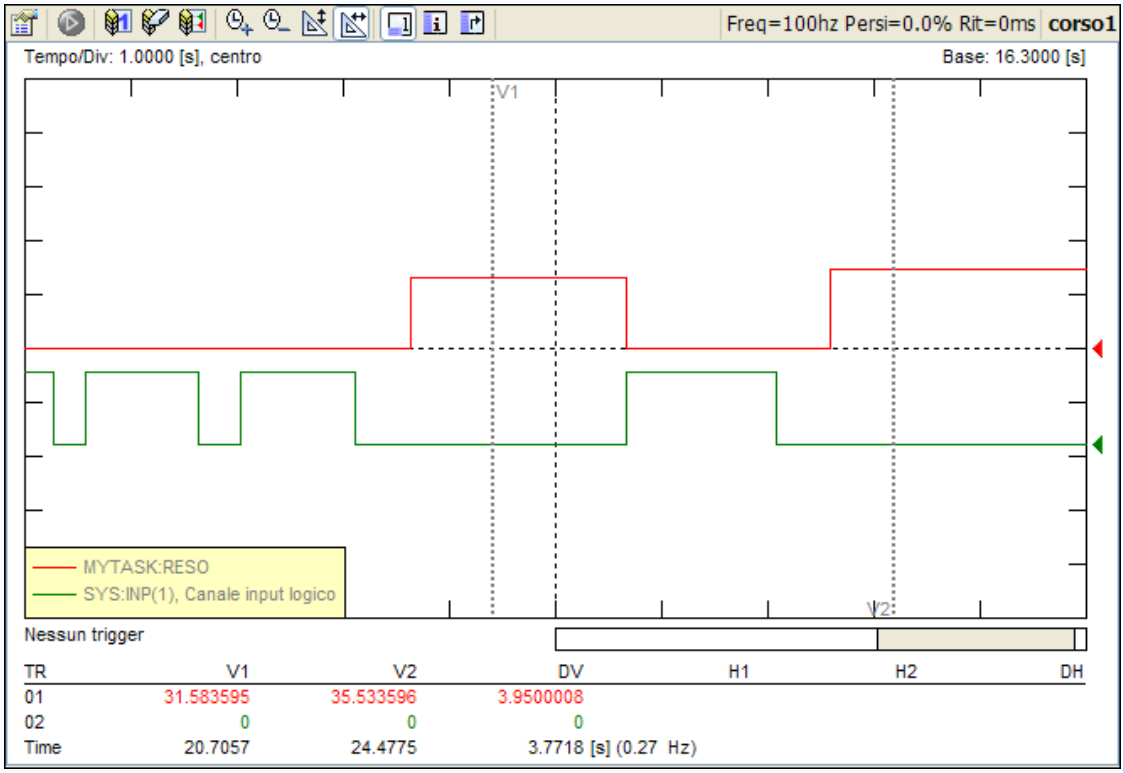
Providing ip(n) as a filter variable, for example, will result in a filter in space, while providing tfb as a filter variable will result in a filter in time (units seconds).

Each fall_filt() function has its own memory of the "condition" state. The front evaluation is done between the current state of the condition and the memory related to the single function call. (e.g., if we have at step 1000 `if(fall_filt(inp(1), tfb, 0.5))` and at step 1100 `if(fall(inp(1), tfb, 0.5))`, although they refer to the same condition, the evaluation is done separately, each according to the state stored by running step 1000 and then step 1100).

Examples

Example of use:

```
reso = fall_filt(inp(1), tfb, 0.5)
```




```
; temporal filter on inp(1)
inp1f = not(fall_filt(rise_filt(inp(1), tfb, TimeFilter), tfb,
TimeFilter))

; calculation of the distance between the input descent and the
current position
; to handle an output between two thresholds
posDw = fall_filt(inp(1), ip(1), spaceFilter)
if(posDw)
    ; we are down with input. we see the space traveled
    deltaPos = abs(ip(1) - posDW)
    ; open hatch to eliminate piece
    out(oApriBotola) = range(deltaPos,  SogliaApriBotola,
SogliaChiusuraBotola)
else
    out(oApriBotola) = false
endif
```

Function filter_lp()

Digital low-pass filter function. Each function call will have its own storage.

Syntax	**real filter_lp (real val, real omega, real valQ)**
val	Is the value (or a generic mathematical expression) to be filtered
omega	Value of the cut filter omega in radians per second (omega = 2 * K_PI 17 * cut frequency (Hz)). This can be a generic mathematical expression
valQ	Value of the filter coefficient Q (Q must be >= 0.75). It can be a generic mathematical expression
Result	Returns the filtered value
Validity	Rule
Note	Automatic omega limitation: (omega_max = 0.15 / si * 2 * K_PI) 17 -> i.e. no more than 15% of omega value corresponding to rule sampling time. (e.g. if Freq = 200Hz -> filter cut frequency <= 30 Hz)

Examples

Example of use:

```
function filtraCV()
    ; We filter the actual speed read from the master encoder
    cVf = filter_lp 317(cv(AX_C), 2.0 * K_PI 17 * rrFreqFilCv, rrQfilt)
end_fun
```

Function filter_lp_st()

Digital low-pass filter function with specific external storage, so it can be used in multiple calls.

Syntax	**real filter_lp_st (real val, real omega, real valQ, real storage[2], real period, i32 flagInit)**
val	This is the value (or a generic mathematical expression) to be filtered out
omega	Value of the cut filter omega in radians per second (omega = 2 * K_PI 17 * cut frequency (Hz)). It can be a generic mathematical expression

© 2025 Robox SpA

Syntax	**real filter_lp_st (real val, real omega, real valQ, real storage[2], real period, i32 flagInit)**
valQ	Value of the filter coefficient Q (Q must be >= 0.75). It can be a generic mathematical expression
storage	Storage variable for mathematical processing of the filter function
period	Sampling interval, if <= 0 si is assumed
flagInit	Variable for filter initialization request; imposing a value different than 0 will reset the filter and the *flagInit* variable will be written to 0. The first time the function is called this variable must have a value different than 0
Result	Returns the filtered value
Validity	Rule
Note	Automatic omega limitation: (omega_max = 0.15 / si * 2 * K_PI) -> i.e. no more than 15% of the omega value corresponding to the rule sampling time. (e.g. if Freq = 200Hz -> filter cut frequency <= 30 Hz)

The parameters are as for filter_lp(), but in addition it allows you to specify a storage variable used for mathematical processing of the filter function.

In this way, filters can also be used with multiple calls (e.g., within for loops).

Examples

Example of use:

```
real cv_filt[MAX_ASSI]
real store[MAX_ASSI][2]
int  flgInit[MAX_ASSI]

for(i=0, i<MAX_ASSI, i=i+1)
    cv_filt[i] = filter_lp_st(cv(i+1), omega, Q, store[i], 0,
flgInit[i])
end_for
```

Function filter_notch()

Digital notch filter function. Each function call will have its own storage.

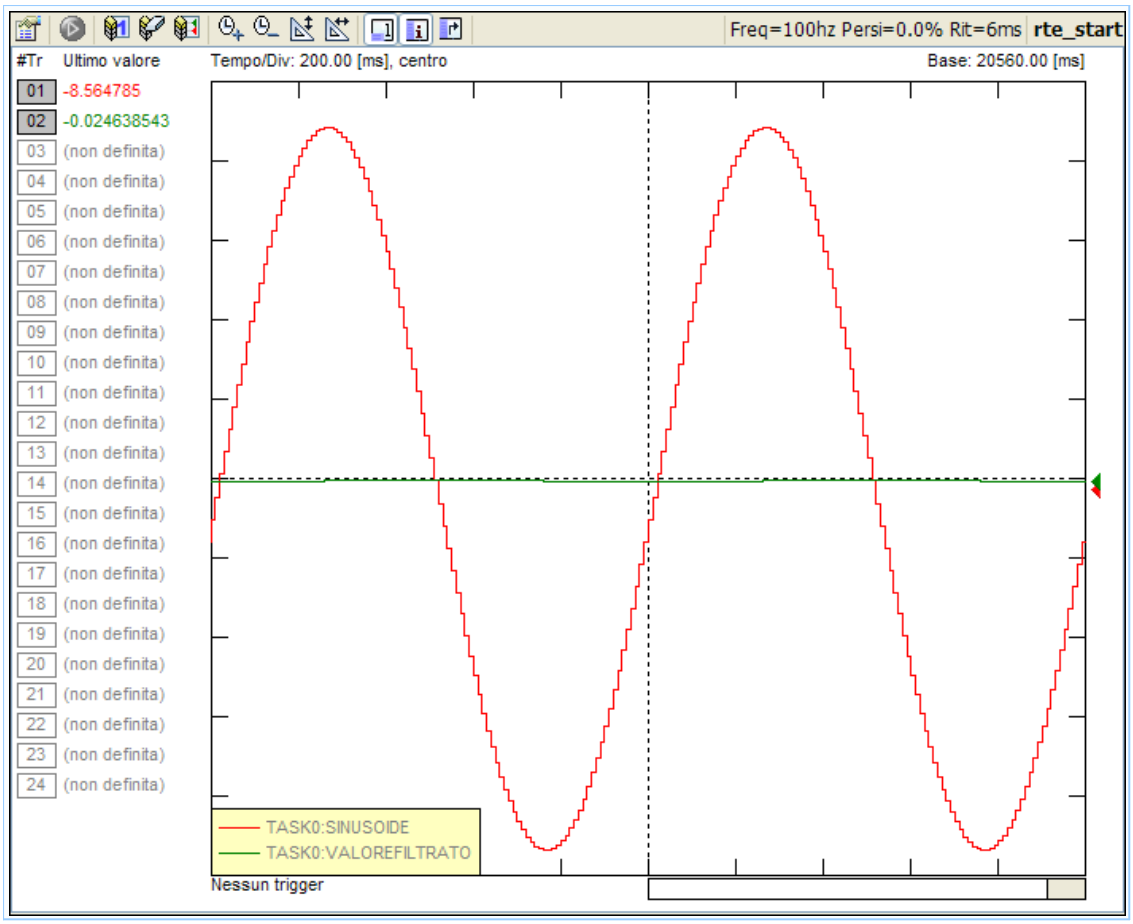
Syntax	**real filter_notch (real val, real omega, real valQ)**
val	Is the value (or a generic mathematical expression) to be filtered
omega	Value of the cut filter omega in radians per second (omega = 2 * K_PI 17 * cut frequency (Hz)). It can be a generic mathematical expression
valQ	Value of the filter Q coefficient (Q must be >= 0.75). Can be a generic mathematical expression
Result	Returns the filtered value
Validity	Rule
Note	Automatic omega limitation: (omega_max = 0.15 / si * 2 * K_PI) 17 -> i.e. no more than 15% of omega value corresponding to rule sampling time. (e.g. if Freq = 200Hz -> filter cut frequency <= 30 Hz)

Examples

Example of use:

```
Rule 2
Axes 1
motion
    ; I construct a sine wave of amplitude 100 and frequency
rrFreqSin
    sinusoide = 100.0 * sin 576 (2 * K_PI 17 * rrFreqSin * tfb)
    ; Let's filter
    valoreFiltrato = filter_notch 319 (sinusoide, 2 * k_pi 17 *
rrFreqNotch , rrQnotch)
end_motion
; (se rrFreqSin = 1 ; rrFreqNotch = 1 ; rrQnotch = 0.8 )
```

© 2025 Robox SpA



Function filter_notch_st()

Digital notch filter function with specific external storage, so it can be used in multiple calls.

Syntax	**real filter_notch_st (real val, real omega, real valQ, real storage[2], real period, i32 flagInit)**
val	Is the value (or a generic mathematical expression) to be filtered
omega	Value of the cut filter omega in radians per second (omega = 2 * K_PI[17] * cut frequency (Hz)). It can be a generic mathematical expression
valQ	Value of the filter Q coefficient (Q must be >= 0.75). Can be a generic mathematical expression
storage	Storage variable for mathematical processing of the filter function
period	Sampling interval, if <= 0 si is assumed

Syntax	**real filter_notch_st (real val, real omega, real valQ, real storage[2], real period, i32 flagInit)**
flagInit	Variable for filter initialization request; imposing a value different than 0 will reset the filter and the *flagInit* variable will be written to 0. The first time the function is called this variable must have a value different than 0
Result	Returns the filtered value
Validity	Rule
Note	Automatic omega limitation: (omega_max = 0.15 / si * 2 * K_PI) 17 -> i.e. no more than 15% of omega value corresponding to rule sampling time. (e.g. if Freq = 200Hz -> filter cut frequency <= 30 Hz)

The parameters are as for filter_notch 319(), but in addition it allows you to specify a storage variable used for mathematical processing of the filter function.

In this way, filters can also be used with multiple calls (e.g., within for loops).

Examples

Example of use:

```
real cv_filt[MAX_ASSI]
real store[MAX_ASSI][2]
int  flgInit[MAX_ASSI]
....
for(i=0, i<MAX_ASSI, i=i+1)
    cv_filt[i] = filter_notch_st 320(cv(i+1), omega, Q, store[i], 0,
flgInit[i])
end_for
```

Function first_time()

This function returns TRUE only during the first execution of the rule in question, after it has been placed in execution by a group 694 instruction or rc(n). That is, if since the previous sampling rc(n), with n executor currently in use, has changed.

Syntax	**i32 first_time ()**
Result	Returns:

Value	Description
<> 0	If first execution of rule
0	Otherwise

Syntax	**i32 first_time ()**
Validity	Rule
Note	It is very useful for performing initializations

Examples

Example of use:

```
rule 1
axes (1)
motion
    if(first_time321())
        fase = 0
        rr(100) = 0
    endif
...
...
...
end_motion
end_rule
```

Function force()

Function to force the state of I/O.

Syntax	**i32 force (what, value)**
what	Object to be forced. Can be:

Type	Description
inp(nCh)	Where *nCh* is the input channel to be forced. Can be any expression
inp_w(nW)	Where *nW* is the index of the input word to be forced. Can be any expression
inp_w(nW).(iB)	Where *nW* is the index of the input word and *iB* is the bit of the input word to be forced. It can be any expression
out(nCh)	Where *nCh* is the output channel to be forced. It can be any expression

© 2025 Robox SpA

Syntax	**i32 force (what, value)**
	<table><tr><th>Type</th><th>Description</th></tr><tr><td>out_w(nW)</td><td>Where *nW* is the index of the output word to be forced. Can be any expression</td></tr><tr><td>out_w(nW).(iB)</td><td>Where *nW* is the index of the output word and *iB* is the bit of the output word to be forced. It can be any expression</td></tr></table>
value	Value to force
Result	Returns: <table><tr><th>Value</th><th>Description</th></tr><tr><td>0</td><td>Operation performed successfully</td></tr><tr><td>-1</td><td>Version of RTE not compatible</td></tr><tr><td>-2</td><td>Invalid index</td></tr><tr><td>-3</td><td>Invalid bit</td></tr></table>
Validity	Rule, Task
Note	Remember that: • Forcing an input means that sw will see the state of the forced input, regardless of the actual state of the physical input • Forcing an output means that the physical output will take the forced value regardless of the desired value imposed by sw

Examples

Example of use:

```
$LIT iw_rem  401                ; remote word input index
...
...
$LIT pbStart  inp_w(iw_rem).(4) ; start pushbutton
...
...
; simulates the rising edge on the pushbutton start input.
esitoF = force 322 (pbStart, 0)
```

```
if(esitoF < 0)
    ...                     ; error diagnostics.
endif
dwell 677 (0.02)
esitoF = force(pbStart, 1)
if(esitoF < 0)
    ...                     ; error diagnostics.
endif
dwell(0.2)
```

Example of use:

```
$LIT iw_base_rem  401                    ; remote word input index
...
...
$LIT pbStop  inp((iw_base_rem-1)*16+5)   ; stop pushbutton
...
...
; simulates a request to stop
esitoF = force(pbStop, 0)
if(esitoF < 0)
    ...                     ; error diagnostics.
endif
dwell(0.1)
esitoF = release 534 (pbStop)
if(esitoF < 0)
    ...                     ; error diagnostics.
endif
...
...
```

Esempio di utilizzo:

```
; forces an output
; r(1000) contains the output channel
; r(1001) contains the value to force
esitoF = force(out(r(1000)), r(1001))
if(esitoF < 0)
    ...                     ; error diagnostics.
endif
...
...
```

Function fract()

Function that returns the fractional part of its argument.

Syntax	**real fract (real value)**
value	Value on which to perform the operation
result	Returns the fractional part of the argument

Syntax	**real fract (real value)**
Validity	Rule, Task
Note	

Examples

Example of use:

```
pippo = fract(12.876)
; after execution pippo contains the value: 0.876
```

Function get_float()

This function returns a float read from a buffer at the specified offset.

Syntax	**float get_float (buff, i32 offset)**
buff	Can be an array, a structure, or a register
offset	Offset in number of bytes you want to read at
Result	Returns the value read
Validity	Rule, Task
Notes	No range check is performed, it is up to the user to check that adding the offset and the data size does not exceed the maximum buffer size

NOTE: If the user wishes to extract a float from the buffer of a structure of type bbc_msguser then he must use the function bcc_get_float().

Examples

Example usage:

Once you have loaded the example program on a control, via a shell you can verify the functioning of get_float(). To do this, set a value `((0-:-127) - sizeof(my_data))` to the `offset` variable, then set a non-zero value to the `set_val` variable and verify that the `my_data` variable is equal to `set_val`.

```
u8 buffer[128]
i32 offset
float set_val
float my_data

; -------- loop begin --------
__MAIN_LOOP__

    ; range check
    if(offset >= 0 and (offset + sizeof(my_data)) < 128)
```

© 2025 Robox SpA

```
        set_float 739(buffer, offset, set_val)
        my_data = get_float 325(buffer, offset)
    endif

END_MAIN_LOOP
```

Function get_hw_info()

This instruction provides information about the boards directly connected to the CPU.

Syntax	**i32 get_hw_info (i32 nSlot , stru_hw_info stru [, i32 info=0])**
nSlot	Slot number present in the rhw.cfg file. **NOTE**: *nSlot* does not correspond to the number of slots that are assigned to Axioline 94 modules.
stru	Structure of type stru_hw_info 841 where the information is saved when the function is called
info	Parameter to request desired or current values. (optional, default 0) Value / Description 1 / Current values are returned 0 / The values in the rhw.cfg file are returned **NOTE**: Values differ only in the case of alarm 988.
Result	Returns: Value / Meaning <> 0 / Successful operation 0 / *nSlot* is not within the expected range. Alarm 9900 is generated
Validity	Task
System Requirements	RTE 34.20.3
Note	The *info* parameter was implemented by RTE 34.27.2

Examples

Example of use:

```
; Parte del file rhw.cfg dove è scritto il numero dello slot di ogni
modulo connesso
SLOT 0 CPURBX
    PSER 1 RS232
    PSER 2 RS485
    ETH 1
    ETH 2
    CAN 1 Unused
    CAN 2 Unused
    PROFIBUS 1 Unused
SLOT 1 AUX1_URMC
SLOT 2 AXL_DRIVER
SLOT 3 AXL_F_SSDO8_3_1F
SLOT 4 AXL_F_SSDI8_4_1F
SLOT 5 AXL_F_LPSDO8_3_1F
SLOT 6 AXL_F_CNT2_INC2

; Task1
$task 1
$include "litreg"

; variables definition
STRU_HW_INFO hw
int result
int nSlot
int inpWoutW
U32 board_code_low
U32 board_code_high
string boardName[64]

nSlot = 1
; start main loop
__MAIN_LOOP__
if(rise(inp(1))) ; ADV button pressed, call function
    result = get_hw_info(nSlot, hw)
    if(result = 1) ; operation OK
        inpWoutW = hw.iwow
        board_code_low = hw.board_code_l
        board_code_high = hw.board_code_h
        boardName = hw.name
    endif
endif
END_MAIN_LOOP
```

© 2025 Robox SpA

Function get_i16()

This function returns a word i16 read from a buffer at the specified offset.

Syntax	**i16 get_i16 (buff, i32 offset)**
buff	Can be an array, structure, or register
offset	Offset in bytes to which you want to read
Result	Returns the value read
Validity	Rule, Task
Note	No range check is performed, it is up to the user to check that adding the offset and the data size does not exceed the maximum buffer size

NOTE: If the user wishes to extract a word from the buffer of a structure of type bbc_msguser 793 then he must use the function bcc_get_i16 198 ().

Examples

Example usage:

Once you have loaded the example program on a control, via a shell you can verify the functioning of get_i16 328 (). To do this, set a value `((0-:-127) - sizeof(my_data))` to the `offset` variable, then set a non-zero value to the `set_val` variable and verify that the `my_data` variable is equal to `set_val`.

```
u8 buffer[128]
i32 offset
float set_val
i16 my_data

; -------- loop begin --------
__MAIN_LOOP__

    ; range check
    if(offset >= 0 and (offset + sizeof(my_data)) < 128)
        set_i16 740 (buffer, offset, set_val)
        my_data = get_i16 328 (buffer, offset)
    endif

END_MAIN_LOOP
```

Function get_i32()

This function returns an i32 read from a buffer at the specified offset.

Syntax	**i32 get_i32 (buff, i32 offset)**
buff	Can be an array, a structure, or a register
offset	Offset in bytes to which you want to read

Syntax	**i32 get_i32 (buff, i32 offset)**
Result	Returns the value read
Validity	Rule, Task
Note	No range check is performed, it is up to the user to check that adding the offset and the data size does not exceed the maximum buffer size

NOTE: If the user wishes to extract a dword from the buffer of a structure of type bbc_msguser 793 then he must use the function bcc_get_i32 198().

Examples

Example usage:

Once you have loaded the example program on a control, via a shell you can verify the functioning of get_i32 328(). To do this, set a value `((0-:-127) - sizeof(my_data))` to the `offset` variable, then set a non-zero value to the `set_val` variable and verify that the `my_data` variable is equal to `set_val`.

```
u8 buffer[128]
i32 offset
float set_val
i32 my_data

; -------- loop begin --------
__MAIN_LOOP__

    ; range check
    if(offset >= 0 and (offset + sizeof(my_data)) < 128)
        set_i32 741(buffer, offset, set_val)
        my_data = get_i32 328(buffer, offset)
    endif

END_MAIN_LOOP
```

Function get_i8()

This function returns a variable i8 read from a buffer at the specified offset.

Syntax	**i8 get_i8 (buff, i32 offset)**
buff	Can be an array, structure, or register
offset	Offset in bytes to which you want to read
Result	Returns the value read
Validity	Rule, Task
Note	No range check is performed, it is up to the user to check that adding the offset and the

© 2025 Robox SpA

Syntax	**i8 get_i8 (buff, i32 offset)**
	data size does not exceed the maximum buffer size

NOTE: If the user wishes to extract a byte from the buffer of a structure of type bbc_msguser 793 then the user must use the function bcc_get_i8 199().

Examples

Example usage:

Once you have loaded the example program on a control, via a shell you can verify the functioning of get_i8 329(). To do this, set a value `((0-:-127) - sizeof(my_data))` to the variable `offset`, then set a non-zero value to the variable `set_val` and verify that the variable `my_data` is equal to `set_val`.

```
u8 buffer[128]
i32 offset
float set_val
i8 my_data

; -------- loop begin --------
__MAIN_LOOP__

    ; range check
    if(offset >= 0 and (offset + sizeof(my_data)) < 128)
        set_i8 743(buffer, offset, set_val)
        my_data = get_i8 329(buffer, offset)
    endif

END_MAIN_LOOP
```

Function get_real()

This function returns a real read from a buffer at the specified offset.

Syntax	**real get_real (buff, i32 offset)**
buff	Can be an array, structure, or register
offset	Offset in bytes to which you want to read
Result	Returns the value read
Validity	Rule, Task
Note	No range check is performed, it is up to the user to check that adding the offset and the data size does not exceed the maximum buffer size

Note: If the user wishes to extract a real from the buffer of a structure of type bbc_msguser 793 then the user must use the bcc_get_double 197() function.

© 2025 Robox SpA

Examples

Example usage:

Once you have loaded the example program on a control, via a shell you can verify the functioning of get_real 330(). To do this, set a value `((0-:-127) - sizeof(my_data))` to the `offset` variable, then impose a non-zero value to the `rr(1)` register and verify that the variable `my_data` is equal to `rr(1)`.

```
u8 buffer[128]
i32 offset
i8 my_data

; -------- loop begin --------
__MAIN_LOOP__

    ; range check
    if(offset >= 0 and (offset + sizeof(my_data)) < 128)
        set_real 744(buffer, offset, rr(1))
        my_data = get_real 330(buffer, offset)
    endif

END_MAIN_LOOP
```

Function get_u16()

This function returns a word u16 read from a buffer at the specified offset.

Syntax	**u16 get_u16 (buff, i32 offset)**
buff	Can be an array, structure, or register
offset	Offset in bytes to which you want to read
Result	Returns the word read
Validity	Rule, Task
Note	No range check is performed, it is up to the user to check that adding the offset and the data size does not exceed the maximum buffer size

NOTE: If the user wishes to extract a word from the buffer of a structure of type bbc_msguser 793 then the user must use the function bcc_get_u16 201().

Examples

Example usage:

Once you have loaded the example program on a control, via a shell you can verify the functioning of get_u16 331(). To do this, set a value `((0-:-127) - sizeof(my_data))` to the `offset` variable, then impose a non-zero value to `inp_w` 200 and verify that the variable `my_data` is equal to `inp_w 200`.

```
u8 buffer[128]
i32 offset
u16 my_data

; -------- loop begin --------
__MAIN_LOOP__

    ; range check
    if(offset >= 0 and (offset + sizeof(my_data)) < 128)
        set_u16 745(buffer, offset, inp_w(200))
        my_data = get_u16 331(buffer, offset)
    endif

END_MAIN_LOOP
```

Function get_u32()

This function returns a u32 variable read from a buffer at the specified offset.

Syntax	**u32 get_u32 (buff, i32 offset)**
buff	Can be an array, structure, or register
offset	Offset in bytes to which you want to read
Result	Returns the value read
Validity	Rule, Task
Note	No range check is performed, it is up to the user to check that adding the offset and the data size does not exceed the maximum buffer size

NOTE: If the user wishes to extract a u32 from the buffer of a structure of type bbc_msguser 793 then the user must use the bcc_get_u32 201() function.

Examples

Example usage:

Once you have loaded the example program on a control, via a shell you can verify the functioning of get_u16 332(). To do this, set a value `((0-:-127) - sizeof(my_data))` to the `offset` variable, then set a non-zero value to `set_val` and verify that the variable `my_data` is equal to `set_val`.

```
u8 buffer[128]
i32 offset
u32 set_val
u32 my_data

; -------- loop begin --------
__MAIN_LOOP__
```

```
    ; range check
    if(offset >= 0 and (offset + sizeof(my_data)) < 128)
        set_u32 746(buffer, offset, set_val)
        my_data = get_u32 332(buffer, offset)
    endif

END_MAIN_LOOP
```

Function get_u8()

This function returns a variable u8 read from a buffer at the specified offset.

Syntax	**u8 get_u8 (buff, i32 offset)**
buff	Can be an array, structure, or register
offset	Offset in bytes to which you want to read
Result	Returns the value read
Validity	Rule, Task
Note	No range check is performed, it is up to the user to check that adding the offset and the data size does not exceed the maximum buffer size

NOTE: If the user wishes to extract a byte from the buffer of a structure of type bbc_msguser 793 then the user must use the function bcc_get_u8 202().

Examples

Example usage:

Once you have loaded the example program on a control, via a shell you can verify the functioning of get_u8 333(). To do this, set a value `((0-:-127) - sizeof(my_data))` to the `offset` variable, then set a non-zero value to `set_val` and verify that the variable `my_data` is equal to `set_val`.

```
u8 buffer[128]
i32 offset
u8 set_val
u8 my_data

; -------- loop begin --------
__MAIN_LOOP__

    ; range check
    if(offset >= 0 and (offset + sizeof(my_data)) < 128)
        set_u8 747(buffer, offset, set_val)
        my_data = get_u8 333(buffer, offset)
    endif

END_MAIN_LOOP
```

© 2025 Robox SpA

Function get_virt_ow()

This function allows reading the real state of a virtual output word. Real state means the state programmed by a software application plus any forcing.

Syntax	**i32 get_virt_ow (i32 outW)**
outW	Number of out_w whose real state you want to read
Result	Returns the value of the actual state of the selected out_w
Validity	Rule, Task
Note	In case of illegal code,alarm 9900 is generated

Function gv_create()

Function to create a new global variable.

Creates the information in memory and, if required, also in retentive memory.

If requested, creates the definition file. Can also create an interface to the global variable.

Syntax	**i32 gv_create (string var, string dataType, i32 len, string wrtPsw, i32 flag, i32 gvID, buff)**
var	Variable name. Can contain letters, numbers, spaces, and special characters. Not case sensitive
dataType	Description of data type. Can contain letters, numbers, spaces and special characters. If omitted (""), the data type will be inserted automatically
len	Size in bytes of the global variable
wrtPsw	Password for write access. If omitted (""), the password is not used
flag	<table><tr><th>HEX value</th><th>Description</th></tr><tr><td>0x0001</td><td>Persistent variable</td></tr><tr><td>0x0002</td><td>(Only if defined as a persistent variable) Overwrites the value of an unlinked persistent variable with the same name and size as the new one</td></tr></table>

© 2025 Robox SpA

Syntax	**i32 gv_create (string var, string dataType, i32 len, string wrtPsw, i32 flag, i32 gvID, buff)**

HEX value	Description
0x0004	(Only if defined as a non-persistent variable) Disables the creation of the configuration file
0x0008	(Only if defined as a non-persistent variable) Disables data in the configuration file
0x0010	(Only if a password is used) Enables write access only to the first linked interface with correct password

gvID	Local variable in which the function inserts the interface-related ID to the global variable. In case of an error, the function inserts 0
buff	Local buffer used as a cache connected to the ID (if any). It is used to update the global variable information during read and write operations. When the function is called, its value indicates the initialization value of the global variable. It must necessarily be the same size as the global variable
Result	Returns:

Value	Meaning
0	Operation performed successfully
-1	Name or size NULL
-2	Global variable handler not enabled
-3	Existing variable (or same key)
-4	Persistent variable already exists (or same key) or

© 2025 Robox SpA

Syntax	**i32 gv_create (string var, string dataType, i32 len, string wrtPsw, i32 flag, i32 gvID, buff)**
	<table><tr><th>Value</th><th>Meaning</th></tr><tr><td></td><td>persistent memory full or hash table full</td></tr><tr><td>-5</td><td>Problem during creation of definition file</td></tr><tr><td>-6</td><td>Cache size is smaller than global variable size. Alarm 9900 is generated</td></tr></table>
Validity	Task
Note	

Esempi

Example of use:

Creating 2 GV:

- global_1, attribute i32, size 4 bytes, no password, persistent, no initialization value
- global_2, attribute undefined, size 2 bytes, password 123, only the first connected interface has write access, initialization value 5

```
loc_var_1 = 0
loc_var_2 = 5
result = gv_create 334("global_1", "INT", 4, "", 0x1, gvID_1,
loc_var_1)
result = gv_create("GLOBAL_2", "", 2, "123", 0x10, gvID_2, loc_var_2)
```

Using `gv_info` from shell:

```
gv_info
Generic device command [GV_INFO]
  1   NAME<GLOBAL_2>  DATATYPE<BYTEARRAY_2>  SIZE<2>  ATTR.<Volatile - PSW on Write - Only one can write>
  2   NAME<global_1>  DATATYPE<INT>  SIZE<4>  ATTR.<Persistent>
 2 GLOBAL VARIABLES found
```

Function gv_datatype()

Reads the type of the global variable passed the ID.

Syntax	**string gv_datatype (i32 gvID)**
gvID	ID of the global variable (obtained from gv_link 342 or gv_create 334)

Syntax	**string gv_datatype (i32 gvID)**
Result	Returns: Value: Meaning "Unhandled": Manager not enabled "Unconnected": Invalid interface "*dataType*": The data type of the global variable
Validity	Rule, Task
Note	

Examples

Example of use:

```
; using data from the Example use of global variables.
...
sr(2) = gv_datatype 336(GVInterf_sum)    ; gets the data type of the
GV... in SR(2) there will be : "struct structValues SUM"
```

Function gv_datatype_byname()

Reads the data type of the global variable.

Syntax	**string gv_datatype_byname (string var)**
var	Name of the variable. Can contain letters, numbers, spaces, and special characters. It is not case sensitive
Result	Returns: Value: Meaning "": The variable does not exist "*dataType*": The data type of the global variable
Validity	Rule, Task
Note	

Examples

Example of use:

```
;using data from the Example use global variables
 ...
sr(2) = gv_datatype_byname(337)("Global structValori raise")    ; gets
the data type of the GV... in SR(2) there will be : "struct
structValori RAISE"
```

Function gv_destroy()

Function that removes the global variable and clears the configuration file.

Syntax	**bool gv_destroy (i32 gvID)**
gvID	ID of the global variable (obtained from gv_link(342) or gv_create(334))
Result	Returns: Value — Meaning TRUE — Operation completed successfully FALSE — Problems during the operation
Validity	Task
Note	

Examples

Example of use:

```
; using data from the Example use global variables
...
esitoGV = gv_destroy(338)(GVInterf_sum)    ; removes the connection.
Subsequent accesses using GVInterf_sum will not work.
if(esitoGV = false)
    alarm_set(151)(5000, 0, 11, esitoGV)
endif
```

Function gv_exist()

This function checks that the specified string matches the name of a defined global variable.

Syntax	**bool gv_exist (string var)**
var	Variable name. Can contain letters, numbers, spaces, and special characters. Not case sensitive
Result	Returns:

© 2025 Robox SpA

Syntax	**bool gv_exist (string var)**
	Value / Meaning TRUE / Variable found FALSE / Variable not found
Validity	Task
Note	

Examples

Example of use:

```
; using data from the Example use global variables.
 ...
esitoGV = gv_exist 338("global_1")    ; check if there is a GV that is
called "global_1"
```

Function gv_flags()

This function returns the flag of the global variable.

Syntax	**i32 gv_flags (i32 gvID)**
gvID	ID of global variable (obtained from gv_link 342 or gv_create 334)
Result	Returns:

Bit	Hex value	Meaning
	If = 0	The variable does not exist
0	0x00000001	Persistent variable (R/W)
1	0x00000002	Overwrites the value of persistent GVs when creating GV from instruction (R/W)
2	0x00000004	Disables creation of configuration file (if not persistent) (R/W)

© 2025 Robox SpA

Syntax	**i32 gv_flags (i32 gvID)**

Bit	Hex value	Meaning
3	0x00000008	Disables data in the configuration file (if not persistent) (R/W)
4	0x00000010	Enables write access only to the first interface linked with correct password (R/W)
16	0x00010000	GV correctly linked to retentive component
17	0x00020000	GV to be linked to the retentive component
18	0x00040000	Write access
24	0x01000000	Internal use - name
25	0x02000000	Internal use - data
26	0x04000000	Internal use - dataType
27	0x08000000	Internal use - wrtPsw

Validity	Rule, Task
Note	

Examples

Example of use:

```
; using data from the Example use global variables
...
r(2) = gv_flags(GVInterf_sum)    ; returns the configuration flag of
the variable
```

Function gv_flags_byname()

This function returns the flag of the global variable.

Syntax	**i32 gv_flags_byname (string var)**
var	Variable name. Can contain letters, numbers, spaces, and special characters. Not case sensitive
Result	Returns:

Bit	Hex value	Meaning
	Se = 0	Variable does not exist
0	0x00000001	Persistent variable (R/W)
1	0x00000002	Overwrites the value of persistent GV when creating GV from instruction (R/W)
2	0x00000004	Disables creation of the configuration file (if not persistent) (R/W)
3	0x00000008	Disables data in the configuration file (if not persistent) (R/W)
4	0x00000010	Enables write access only to the first interface linked with correct password (R/W)
16	0x00010000	GV correctly linked to retentive component
17	0x00020000	GV to be linked to the retentive component

© 2025 Robox SpA

Syntax	**i32 gv_flags_byname (string var)**
	<table><tr><th>Bit</th><th>Hex value</th><th>Meaning</th></tr><tr><td>18</td><td>0x00040000</td><td>Write access</td></tr><tr><td>24</td><td>0x01000000</td><td>Internal use - name</td></tr><tr><td>25</td><td>0x02000000</td><td>Internal use - data</td></tr><tr><td>26</td><td>0x04000000</td><td>Internal use - dataType</td></tr><tr><td>27</td><td>0x08000000</td><td>Internal use - wrtPsw</td></tr></table>
Validity	Rule, Task
Note	

Examples

Example of use:

```
; using data from the Example I use global variables
...
r(2) = gv_flags_byname 341("Global Real 1")    ; returns the
configuration flag of the variable
```

Function gv_link()

Function to link a global variable.

Creates an interface to the variable. The specified local variable is closely associated with the interface.

Sintassi	**i32 gv_link (string var, i32 len, string wrtPsw, i32 gvID, buff)**
var	Variable name. Can contain letters, numbers, spaces, and special characters. Not case sensitive
len	Size in bytes of the global variable
wrtPsw	Password for write access. If omitted (""), the password is not used
gvID	Local variable in which the function inserts the ID relative to the interface to the global variable. In case of an error, the function inserts 0

© 2025 Robox SpA

Sintassi	**i32 gv_link (string var, i32 len, string wrtPsw, i32 gvID, buff)**
buff	Local buffer used as a cache connected to the ID (if present). It is used to update the global variable information during read and write operations. When the function is called, its value indicates the initialization value of the global variable. It must necessarily be the same size as the global variable
Result	Returns:

Value	Meaning
0	Operation successfully completed
1	Link created, but GV is in lock, so 'locVar' is not updated
2	Link created, but no write access (wrong password or not the first link)
3	Link created, but GV is in lock, so 'locVar' is not updated and you do not have write access (wrong password or it is not the first link)
-1	Variable does not exist
-2	Invalid size
-3	Invalid password
-10	Manager not enabled
-11	NULL name
-12	locVar NULL
-13	gvIDrule NULL
-14	size NULL
-15	The size of the cache is smaller than the size of the variable. Alarm 9900 is generated

© 2025 Robox SpA

Sintassi	**i32 gv_link (string var, i32 len, string wrtPsw, i32 gvID, buff)**
Validity	Rule, Task
Note	

Function gv_lock()

Function that blocks access to the global variable.

Access to the GV is not allowed by any other interface.

Syntax	**bool gv_lock (i32 gvID)**
gvID	ID of the global variable (obtained from gv_link 342 or gv_create 334)
Result	Returns: Value — Meaning TRUE — Variable locked FALSE — Variable already locked (by another interface or by itself)
Validity	Rule, Task
Note	

Function gv_name()

Function that returns the name of the global variable.

Syntax	**string gv_name (i32 gvID)**
gvID	ID of the global variable (obtained from gv_link 342 or gv_create 334)
Result	Returns: Value — Meaning "Unhandled" — Manager not enabled "Unconnected" — Invalid interface "*vName*" — The name of the global variable
Validity	Rule, Task
Note	

© 2025 Robox SpA

Examples

Example of use:

```
; using data from the Example use global variables
...
sr(1) = gv_name(GVInterf_sum)    ; gets the name of the GV--in SR(1)
there will be : "Global structValori sum"
```

Function gv_read()

This function updates the local variable data (provided by gv_link or gv_create) associated via the interface, reading from the GV.

Syntax	**i32 gv_read (i32 gvID)**
gvID	ID of the global variable (obtained from gv_link or gv_create)
Result	Returns:

Value	Meaning
0	Operation successfully completed
-1	Manager not enabled
-2	invalid*gvID* (nonexistent interface)
-3	GV is locked

Validity	Rule, Task
Note	

Function gv_read_partial()

This function partially updates the local variable data (provided by GV_LINK or GV_CREATE) associated via the interface, reading from the GV.

Syntax	**i32 gv_read_partial (i32 gvID, u32 offset, u32 dataLen)**
gvID	ID of the global variable (obtained from gv_link or gv_create)
offset	Offset in bytes in the variable (from 1 to the size of GV-1)
dataLen	Length of the data to be updated (from 1 to the size of GV)

© 2025 Robox SpA

Syntax	**i32 gv_read_partial (i32 gvID, u32 offset, u32 dataLen)**
Result	Returns:

Value	Meaning
0	Operation successfully completed
-1	Manager not enabled
-2	invalid*gvID* (nonexistent interface)
-3	GV is locked

Validity	Rule, Task
Note	

Function gv_size()

This function returns the size of the global variable.

Syntax	**i32 gv_size (i32 gvID)**
gvID	ID of the global variable (obtained from gv_link 342 or gv_create 334)
Result	Returns:

Value	Meaning
0	Global variable handler not enabled or invalid interface
size	Size of GV

Validity	Rule, Task
Note	

Examples

Example of use:

```
; using data from the Example use global variables
...
r(2) = gv_size(GVInterf_real)    ; gets the size of the GV, in R(2)
is set 8
```

Function gv_size_byname()

This function returns the size of the global variable.

Syntax	**i32 gv_size_byname (string var)**
var	Variable name. Can contain letters, numbers, spaces, and special characters. Not case sensitive
Result	Returns: Value: 0 — The variable does not exist *size* — Size of the GV
Validity	Rule, Task
Note	

Examples

Example of use:

```
; using data from the Example use of global variables
...
r(2) = gv_size_byname(347)("Global Real 1")    ; gets the size of the GV,
in R(2) is set 8
```

Function gv_unlink()

This function removes the connection of the interface with the global variable.

Syntax	**bool gv_unlink (i32 gvID)**
gvID	ID of the global variable (obtained from gv_link(342) or gv_create(334))
Result	Returns: Value: TRUE — Connection removed FALSE — Operation failed
Validity	Rule, Task
Note	

Examples

© 2025 Robox SpA

Example of use:

```
; using data from the Example use of global variables
...
esitoGV = gv_unlink(GVInterf_sum)   ; removes the connection.
Subsequent operations using GVInterf_sum will no longer be valid
```

Function gv_unlock()

Function that unlocks access to the global variable.

Access to the GV is now allowed to all other interfaces.

Syntax	**bool gv_unlock (i32 gvID)**
gvID	ID of the global variable (obtained from gv_link or gv_create)
Result	Returns: Value: TRUE — Meaning: Variable unlocked Value: FALSE — Meaning: Variable already unlocked (or locked by another interface)
Validity	Rule, Task
Note	

Function gv_update_file()

This function updates the global variables configuration file.

Syntax	**bool gv_update_file (i32 gvID)**
gvID	ID of the global variable (obtained from gv_link or gv_create)
Result	Returns: Value: TRUE — Meaning: Operation performed successfully. Value: FALSE — Meaning: The operation had problem
Validity	Task
Note	

© 2025 Robox SpA

Examples

Example of use:

```
; using data from the Example use of global variables
...
esitoGV = gv_update_file(GVInterf_raise)    ; update the
configuration file with the current values
if(esitoGV = false)
    alarm_set(5000, 0, 12, esitoGV)
endif
```

Function gv_write()

This function updates the value of the global variable with the value of the associated local variable via the interface ('*locVar*' provided by gv_link or gv_create).

Syntax	**i32 gv_write (i32 gvID)**
gvID	ID of the global variable (obtained from gv_link or gv_create)
Result	Returns:

Value	Meaning
0	Operation successfully completed
-1	Manager not enabled
-2	Invalid*gvID* (nonexistent interface)
-3	GV is blocked
-4	Write access blocked by password

Validity	Rule, Task
Note	

Function gv_write_partial()

This function partially updates the value of the global variable with the value of the associated local variable via the interface ('*locVar*' provided by gv_link or gv_create).

Syntax	**i32 gv_write_partial (i32 gvID, u32 offset, u32 dataLen)**
gvID	ID of the global variable (obtained from gv_link or gv_create)

© 2025 Robox SpA

Syntax	**i32 gv_write_partial (i32 gvID, u32 offset, u32 dataLen)**
offset	Offset in bytes in the variable (from 1 to the size of GV-1)
dataLen	Length of the data to be updated (from 1 to the size of GV)
Result	Returns:

Value	Meaning
0	Operation successfully completed
-1	Manager not enabled
-2	Invalid*gvID* (nonexistent interface)
-3	GV is blocked
-4	Write access blocked by password
-5	Invalid offset or data length

Validity	Rule, Task
Note	

Function in_range()

Function that returns the incoming value appropriately limited within the specified range.

Syntax	**real in_range (real value, real min, real max)**
value	Real unit to be examined (can be a generic real expression)
min	Lower limit value (can be a generic real expression)
max	Upper limit value (can be a generic real expression)
Result	Returns:

Condition	Value
value > max	*max*
value < min	*min*

© 2025 Robox SpA

Syntax	**real in_range (real value, real min, real max)**
	Condition: Otherwise — Value: *value*
Validity	Rule, Task
Note	With the RC3E compiler, it is good to specify the cast of the input data type in order to avoid warnings during compilation. (e.g. `in_range((real) rrParPuntiCammaLMot, (real)2.0, (real)max_punti_camma_lmot))`

Examples

Example of use:

```
lenObj = in_range(rr(1), 10, 100)
lenObj = in_range(rr(1), rr(2), max_str(1) - ip(1))
```

Function inc()

Increases the variable of the specified units and also returns its new content.

Syntax	**real inc (real var [, real value=1])**
var	Value to be incremented
value	Increment value (optional, default 1)
result	Returns the new content of *var* (incremented value: *var+value*)
Validity	Rule, Task
Note	The result can also be omitted

Examples

Example of use:

```
if(inp(1))
    inc(rr(1), 10)
else
    dec(rr(1),10)
endif

real count1, count2
if(inp(1))
    count1 = inc(rr(1),10)
```

© 2025 Robox SpA

```
else
    count2 = dec(rr(1),10)
endif

for(i=8, i > 0, inc(i)) ; corresponds to for(i=8, i>0, i=i+1)

appo = inc(pluto, 10)
; corresponds to:
; pluto = pluto +10
; appo  = pluto

inc(r(1))
; corresponds to r(1) = r(1) + 1
```

Function index_of()

Function that returns the index of the resource passed as an argument.

The function is able to distinguish associations to:

r(index), rr(index), sr(index), nvr(index), nvrr(index), nvsr(index), inp(index), out(index), inp_w(index), out_w(index).

Syntax	**i32 index_of (define)**
define	Is the symbolic associated with the variable
Result	Returns the index of the passed argument
Validity	Rule, Task
Note	

The function is actually a compiler macro which, having an argument r/rr/sr/nvr/nvsr/inp/out/inp_w/out_w(index), returns compilation-time <index> instead of index_of(...).

Examples

Example of use:

```
lit rrRicettaAppo  rr(345)
lit iAriaOk        inp(264)
...
...
idxPar = index_of 352(rrRicettaAppo)   ; idxPar assume the value 345
idxInp = index_of(iAriaOk)          ; idxInp assume the value 264
```

© 2025 Robox SpA

Function ipaddr_to_str()

Conversion from IP address represented in an integer to a string.

Syntax	**i32 ipaddr_to_str (u32 addr, string sAddr)**
addr	Parameter containing the address to be converted
sAddr	Parameter in which the function writes the converted address
Result	Returns: Value: Meaning <> 0: Operation successfully completed 0: *sAddr* is 0 or string size is too small
Validity	Rule, Task
Note	

The string has the format: "xxx.xxx.xxx.xxx."

The integer is of type u32 with format: u32 *addr* = 0xaabbccdd; aa, bb, cc and dd represent the 4 components of the IP address expressed in hexadecimal format.

Examples

Example of use:

```
int reso
string addrSr
u32 addrNum
reso = ipaddr_to_str(addrNum, addrSr)
; if addrNum= 0xc0a8010a --> addrSr = "192.168.1.10"
; if addrNum= 0x0a001110 --> addrSr = "10.0.17.16"
```

Function is_alarm_par_present()

This function allows you to check if there is an alarm in the alarm stack (32 locations) by also specifying the value of one or more parameters.

Syntax	**i32 is_alarm_par_present (i32 nAl, i32 nAx, u32 parMask, real par1 [, real par2=0] [, real par3=0])**
nAl	Alarm number to check for (1-:-9999)
nAx	Axis number in alarm (1-:-32) or 0 if non-axis alarm
parMask	Mask to indicate which parameters are affected:

© 2025 Robox SpA

Syntax	**i32 is_alarm_par_present (i32 nAl, i32 nAx, u32 parMask, real par1 [, real par2=0] [, real par3=0])**
	<table><tr><th>Value</th><th>Description</th></tr><tr><td>0x0</td><td>None. Use is_alarm_present 357()</td></tr><tr><td>0x1</td><td>par1</td></tr><tr><td>0x2</td><td>par2</td></tr><tr><td>0x3</td><td>par1 and par2</td></tr><tr><td>0x4</td><td>par3</td></tr><tr><td>0x5</td><td>par1 and par3</td></tr><tr><td>0x6</td><td>par2 and par3</td></tr><tr><td>0x7</td><td>par1, par2 and par3</td></tr></table>
par1	Value of the first alarm parameter
par2	Value of the second alarm parameter. (optional, default 0)
par3	Value of the third alarm parameter. (optional, default 0)
Result	Returns: <table><tr><th>Value</th><th>Description</th></tr><tr><td><> 0</td><td>Alarm present</td></tr><tr><td>0</td><td>Otherwise</td></tr></table>
Validity	Rule, Task
Note	For parameters in the middle that were not specified in the search, a value must still be necessarily entered (indifferent value, even different from the real value). From RTE 34.30.0 if a parameter is passed that was not expected by the alarm, it is still considered as if it were diversity (return 0).

Examples

Example of use:

```
=================================================================
In the user alarm file .txt:
```

```
1010 37 10 0x32000000 "Major alarm: %d, %d, %d" -4 -4 -4     ;
ALM_MAJOR_TEST
```

==========

In litconst.i3:

```
lit ALM_MAJOR_TEST      1010
```

==========

In r3 task:

```
int result, resOK, resErr
...

if(rise 536(rGenerateAlarm))
    result = alarm_set 151(ALM_MAJOR_TEST, 0, 1, 2, 3) ; par1=1,
par2=2, par3=3
    ; In alarm stack will be shown: "Major alarm: 1, 2, 3"
endif

if(result)
; Correct result (return 1)
    resOK = is_alarm_par_present(ALM_MAJOR_TEST, 0, 0x1, 1)    ; par1
    resOK = is_alarm_par_present(ALM_MAJOR_TEST, 0, 0x2, 0, 2)       ;
par2
    resOK = is_alarm_par_present(ALM_MAJOR_TEST, 0, 0x3, 1, 2, 0)   ;
par1 + par2
    resOK = is_alarm_par_present(ALM_MAJOR_TEST, 0, 0x4, 0, 0, 3)   ;
par3
    resOK = is_alarm_par_present(ALM_MAJOR_TEST, 0, 0x5, 1, 0, 3)   ;
par1 + par3
    resOK = is_alarm_par_present(ALM_MAJOR_TEST, 0, 0x6, 0, 2, 3)   ;
par2 + par3
    resOK = is_alarm_par_present(ALM_MAJOR_TEST, 0, 0x7, 1, 2, 3)   ;
par1 + par2 + par3
    ; you can enter any value for middle parameters that are out of
interest
    resOK = is_alarm_par_present(ALM_MAJOR_TEST, 0, 2, -1, 2)
    resOK = is_alarm_par_present(ALM_MAJOR_TEST, 0, 4, 123, 3, 3)
    resOK = is_alarm_par_present(ALM_MAJOR_TEST, 0, 2, 0, 2)

; Wrong result (return 0)
    resErr = is_alarm_par_present(ALM_MAJOR_TEST, 0, 0x1, -1)   ; par1
!= 1
    resErr = is_alarm_par_present(ALM_MAJOR_TEST, 0, 0x2,  2)   ;
error, par1 not inserted
endif
...
```

© 2025 Robox SpA

Function is_alarm_par_present_us()

This function allows you to check if there is an alarm in the alarm stack (32 locations) by also specifying the value of one or more USV parameters.

Syntax	**bool is_alarm_par_present_us (i32 nAl, i32 nAx, u32 parMask [, string s1=0] [, string s2=0] [, string s3=0])**
nAl	Alarm number whose presence to check (0 or 1-:-9999)
nAx	Axis number in alarm (1-:-32) or 0 if non-axis alarm
parMask	Mask to indicate which parameters are affected: Value / Description 0x0 / No USV parameter 0x1 / s1 0x2 / s2 0x3 / s1 and s2 0x4 / s3 0x5 / s1 and s3 0x6 / s2 and s3 0x7 / s1, s2, and s3
s1	User string to be used as the first parameter of the alarm. (optional, default 0)
s2	User string to be used as the second parameter of the alarm. (optional, default 0)
s3	User string to be used as the third parameter of the alarm. (optional, default 0)
Result	Returns: Value / Description TRUE / Alarm present FALSE / • alarm not present • illegal alarm number • alarm does not have USV and parMask different from 0
Validity	Rule, Task
System Requirements	From RTE 34.30.0

© 2025 Robox SpA

Syntax	**bool is_alarm_par_present_us (i32 nAl, i32 nAx, u32 parMask [, string s1=0] [, string s2=0] [, string s3=0])**
Note	

Diagnostics

- Generates 9986 wrong test_alarm presence funct. on alarm:xxx if alarm does not have a defined USV and parMask other than 0 (where xxx alarm number)

Examples

Example usage:

```
string s1
string s2
int result
...
; test presence of axis mayor alarm
str_format (s1, "Broken Brake")
result = is_alarm_par_presence_us(1101, 3, 1, s1)
...
; test presence of one warning about Emergency push button pressed
str_format (s1, "Emergency Pressed")
result = is_alarm_par_presence_us(4101, 0, 1, s1)
...
; test presence of a specific warning about Emergency push button
pressed
str_format (s2, "near first door")
result = is_alarm_par_presence_us(4101, 0, 2, 0, s2)

; alarm definition in user file
1101 -1 14 0x22001000 "Fault <%s>" 17 0 0
4101 53 24 0x22001000 "Warning <%s:%s>" 17 17 0
```

Function is_alarm_present()

This function allows you to check if there is an alarm in the alarm stack (32 positions).

Syntax	**i32 is_alarm_present ([i32 nAl=0] [, i32 nAx=0])**
nAl	Alarm number whose presence to check (1-:-9999). (optional, default 0)
nAx	Axis number in alarm (1-:-32) or 0 if non-axis alarm. (optional, default 0)
Result	Returns:

© 2025 Robox SpA

Syntax	**i32 is_alarm_present ([i32 nAl=0] [, i32 nAx=0])**
	Value: <> 0 — Alarm present Value: 0 — Otherwise
Validity	Rule, Task
Note	

Examples

Example of use:

```
int nAl, nAx, isAlPres
u32 parMask
real pAl1, pAl2, pAl3
...
...
...
if(is_alarm_present(11, 1))
    sr(1) = "axes 1 in following error"
```

Function is_forced()

Function to check if I/O status is forced.

Syntax	**i32 is_forced (what [, i16|i32|u16|u32 sts=NULL [, i32|u32 mask=NULL]])**
what	I/O to be controlled. It can be:

Type	Description
inp(nCh)	Where *nCh* is the input channel. It can be any expression
inp_w(nW)	Where *nW* is the index of the input word. Can be any expression
inp_w(nW).(iB)	Where *nW* is the index of the input word and *iB* is the bit of the input word. It can be any expression
out(nCh)	Where *nCh* is the output channel. Can be any expression

© 2025 Robox SpA

Syntax	**i32 is_forced (what [, i16|i32|u16|u32 sts=NULL [, i32|u32 mask=NULL]])**
	<table><tr><th>Type</th><th>Description</th></tr><tr><td>out_w(nW)</td><td>Where *nW* is the index of the output word. Can be any expression</td></tr><tr><td>out_w(nW).(iB)</td><td>Where *nW* is the index of the output word and *iB* is the bit of the output word. It can be any expression</td></tr></table>
sts	The function will write to *sts* the state of forcing of the word bits: • the X bit of *sts* is 0 if the X bit of the word is forced to FALSE or unforced • the X bit of *sts* is 1 if the X bit of the word is forced to TRUE **NOTE**: Using *sts* as i32 for a word that is considered i16, sign extension will be done on the 32 bits. (optional, default NULL)
mask	The function will write the mask of the forced word bits into *mask*: • the X bit of *mask* is 0 if the X bit of the word is not forced • the X bit of *mask* is 1 if the X bit of the word is forced (either to TRUE or FALSE). Must be u32 if *sts* is i32 or u32, or u16 if *sts* is i16 or u16. (optional, default NULL)
Result	Returns: <table><tr><th>Value</th><th>Description</th></tr><tr><td>1</td><td>Forced entity</td></tr><tr><td>0</td><td>Unforced entity</td></tr><tr><td>-1</td><td>RTE version not compatible</td></tr><tr><td>-2</td><td>Invalid index</td></tr><tr><td>-3</td><td>Invalid bit</td></tr></table>
Validity	Rule, Task

© 2025 Robox SpA

Syntax	**i32 is_forced (what [, i16|i32|u16|u32 sts=NULL [, i32|u32 mask=NULL]])**
Note	It should be remembered that: • forcing an input means that sw will see the state of the forced input, regardless of the actual state of the physical input • forcing an output means that the physical output will take the forced value regardless of the desired value imposed by sw

Examples

Example of use:

```
int result
u16 sts
u16 mask
....
result = release(inp_w(401))
result = force(inp_w(401).0, 1)   ; force inp_w 401.0 to TRUE
result = force(inp_w(401).4, 0)   ; force inp_w 401.4 to FALSE
result = force(inp_w(401).5, 1)   ; force inp_w 401.5 to TRUE
result = force(inp_w(401).12, 0)  ; force inp_w 401.12 to FALSE

result = is_forced(inp_w(401), sts, mask)
; I get:
;     result = 1,     inp_w forced
;     sts = 0x0021,   bit 0/bit 5 forced to TRUE
;     mask = 0x1031, bit 0/bit 4/bit 5/bit 12 forced
```

Example of use:

```
$LIT iw_rem    401                    ; remote word input index
$LIT pbStart   inp_w(iw_rem).(4)      ; start pushbutton
$LIT pbStop    inp((iw_rem-1)*16+5)   ; stop pushbutton
int result
int result2
int result3
...
...
if(is_forced(pbStart) > 0)
    result = release(pbStart)
    if(result < 0)
        ...     ; error diagnostics
    endif
endif

result2 = is_forced(pbStop)
if(result2 < 0)
    ...     ; error diagnostics
endif
```

© 2025 Robox SpA

```
result3 = is_forced(out(r(1000)))   ; r(1000) contains the output
channel
if(result3 < 0)
    ...     ; error diagnostics
endif
```

Function kin_conv()

Function that calculates the acceleration and jerk values that allow, starting from a standstill, to reach *spe* speed in time *tAcc.*

Syntax	**i32 kin_conv (real spe, real tAcc, real pJerk, real rAcc, real rJerk [, real maxAcc=0.0])**
spe	Velocity value to be reached starting from 0. Its absolute value [unit/sec] is used
tAcc	Time to go from 0 to *spe* [sec]
pJerk	Value between 0 and 1 that determines how much jerk to apply to the acceleration segment: • 0, strong stepped acceleration (jerk not used) • 1, gentle triangle acceleration (fully jerked segment - speed ramp to S)
rAcc	Variable in which the required acceleration value is written [units/sec^2]
rJerk	Variable in which the jerk value needed is written [units/sec^3]
maxAcc	Maximum value the acceleration can take. (optional, default 0.0) Its absolute value is used
Result	Returns:

Value	Description
3	*tAcc* and *pJerk* not met due to *maxAcc* limitation,(in which case) the parameter *pJerk* is taken as if it were 0
2	*tAcc* not met due to limitation of *maxAcc*

© 2025 Robox SpA

Syntax	**i32 kin_conv (real spe, real tAcc, real pJerk, real rAcc, real rJerk [, real maxAcc=0.0])**
	<table><tr><th>Value</th><th>Description</th></tr><tr><td>1</td><td>*pJerk* not met due to limitation of *maxAcc*</td></tr><tr><td>0</td><td>All OK</td></tr><tr><td>-2</td><td>Negative or null *tAcc* (assumes 1)</td></tr><tr><td>-3</td><td>*pJerk* outside the range 0-:-1 (assumes 0)</td></tr></table>
Validity	Rule, Task
Note	In the case of acceleration limited by *maxAcc*, when possible, the function still tries to guarantee the duration of the acceleration segment *tAcc* at the expense of the jerk factor *pJerk*

Function log()

Function that returns the natural logarithm of *value*.

Syntax	**real log (real value)**
value	Logarithm argument
Result	Returns the natural logarithm of *value*
Validity	Rule, Task
Note	

Examples

Example of use:

```
reso = log(100)   ; reso = 4.605170185988092
reso = log(1000)  ; reso = 6.907755278982137
reso = log(1)     ; reso = 0
reso = log(0)     ; reso = -inf
```

© 2025 Robox SpA

Function log10()

Function that returns the base-10 logarithm of *value*.

Syntax	**real log10 (real value)**
value	Logarithm argument
Result	Returns the base-10 logarithm of *value*
Validity	Rule, Task
Note	

Examples

Example of use:

```
reso = log10(100)   ; reso = 2
reso = log10(1000)  ; reso = 3
reso = log10(1)     ; reso = 0
reso = log10(0)     ; reso = -inf
reso = log10(234)   ; reso = 2.369215857410143
```

Function loop_time()

This function is used to measure the duration of a software loop.

Syntax	**real loop_time ()**
Result	Returns the time [ms] elapsed between two successive executions of the same function
Validity	Rule, Task
Note	The first time it is executed, the function returns 0 and stores the current time. In case loop_time() is used in a user R3 function, it calculates the time between function calls

Each loop_time() function has its own runtime memory. The duration calculation is the difference between the current time and the memory related to the single function call.

Examples

Example of use:

```
__main_loop__
    time = loop_time()  ; duration between calls of the main
    ...
    ...
end_main_loop
```

© 2025 Robox SpA

Function master_filter()

This function is used to filter the speed of a master. Unlike the filter_lp 317() function it retrieves the position error introduced by the filter.

Syntax	**i32 master_filter (stru_masterF stru [, i32 nAx=0] [, real period=0])**
stru	Structure of type stru_masterF 845 containing input/output parameters
nAx	Master axis number if real axis or 0 if virtual axis. (optional, default 0)
period	Time interval between two consecutive executions of this function. If 0 is assumed to be equal to SI. (optional, default 0)
Result	Returns:

Value	Meaning
-5 [MV_ILLEGAL_ARGS]	Illegal parameters
5 [MV_FOLLOWING]	Tracking in progress
6 [MV_REACHED]	Mission OK

Validity	Rule
Note	

Examples

Example of use:

```
stru_masterf 845 masterFiltered
; We filter the actual speed read from the master encoder 23
result = master_filter 364(masterFiltered, 23, si)
; or
result = master_filter(masterFiltered, 23)
```

Function maximum()

Function that returns the greater of the two values given as input.

Syntax	**real maximum (real val1, real val2)**
val1	First value to compare
val2	Second value to compare
Result	Returns the greater of *val1* and *val2*
Validity	Rule, Task

© 2025 Robox SpA

Syntax	**real maximum (real val1, real val2)**
Note	

Examples

Example of use:

```
velCalc = 60
velMin = 30


vel = maximum(velCalc, velMin)
; vel = 60
```

Function mbc_connect()

Function used to connect the Modbus client to the specified Modbus TCP server and configure the initial index for Modbus registers.

Syntax	**i32 mbc_connect (i32 mbc, const char* ipAddr, i32 port [, i32 baseIdx=-1])**
mbc	Handle Modbus returned by the previous function mbc_init()
ipAddr	IP address of the server to connect to
port	TCP port to be used for the connection. Typically 502
baseIdx	Parameter to determine how to handle the initial entity index. (optional, default -1) Value / Meaning < 0 / Sys_cfg.5 is used to determine how to handle the initial entity index 0 / The initial entity index is 0 (non-standard) > 0 / The initial entity index is 1 (standard)
Result	Returns: Value / Meaning > 0 / Connection with server enabled -1 / Illegal arguments

Syntax	**i32 mbc_connect (i32 mbc, const char* ipAddr, i32 port [, i32 baseIdx=-1])**
	Value: -2 Meaning: Client already connected to a server
Validity	Task
System Requirements	From RTE 34.30.0
Note	

Function mbc_init()

Function used for opening a Modbus/TCP client-type communication.

The function creates a Modbus client that remains alive as long as the current task is running. If the task is restarted, the client is deleted. Therefore, for proper functioning, it is recommended to run the function each time the task is restarted (see restarted variable).

Syntax	**i32 mbc_init (string strIp, i32 nPort, real timeout, real expTime, i32 buffDim [, i32 initCode<0])**
strIp	String with target IP address
nPort	Target port number
timeout	Timeout in seconds on command execution
expTime	Minimum guaranteed time, in seconds, that the response remains in the buffer before being discarded to make place for new messages if the Modbus message handling buffer is full
buffDim	Size of the Modbus message management buffer. Represents the maximum number of commands (Modbus read/write functions) that can be used simultaneously
initCode	Indicates the initial index of units: • <0, the sys_cfg.5 bit is used to determine the initial index of units • 0, the initial index of units is 0 (nonstandard value) regardless of the status of sys_cfg.5 • >0, the initial index of units is 1 (standard value) regardless of the state of sys_cfg.5 (optional, default <0)
Result	Returns the communication management handle that uniquely identifies the

Syntax	**i32 mbc_init (string strIp, i32 nPort, real timeout, real expTime, i32 buffDim [, i32 initCode<0])**
	communication. It is used in all other Modbus management functions. Value / Meaning > 0 / Reference MBC code -1 / Illegal IP address -2 / Illegal port number -3 / Unable to create client
Validity	Task
Note	The function is blocking. It exits only when the operation is completed

Since RTE 34.30.0, a new management has been introduced. The Modbus client is created, but not connected, the management of the connection will then be manageable through the mbc_connect 365() and mbc_shutdown 384() commands. Automatic reconnection to the server is not available for these types of clients.

Syntax	**i32 mbc_init (real timeout, real expTime, i32 buffDim)**
timeout	Timeout in seconds on command execution
expTime	Minimum guaranteed time, in seconds, that the response remains in the buffer before being discarded to make room for new messages if the Modbus message handler buffer is full
buffDim	Size of the Modbus message handling buffer. Represents the maximum number of commands (Modbus read/write functions) that can be used simultaneously
result	Returns the communication management handle that uniquely identifies the communication. It is used in all other Modbus management functions. Value / Meaning > 0 / Reference MBC code -1 / Illegal arguments -3 / Unable to create client

© 2025 Robox SpA

Syntax	**i32 mbc_init (real timeout, real expTime, i32 buffDim)**
Validity	Task
System Requirements	From RTE 34.30.0
Note	

Examples

Example of use:

```
...
; for MODBUS client
lit MODBUS_IP_ADDR       "192.168.0.12"
lit MODBUS_PORT          502
....
....
int mbc
....
....
select(comPhase)
    case 1
        mbc = mbc_init(MODBUS_IP_ADDR, MODBUS_PORT, 2.0, 2.0, 2, 1)
        if(mbc <= 0)
            alarm_set(ALM_BAD_MBC_INIT, 0, mbc)
            comPhase = -1
        else
            comPhase = 2
        endif
        break
    case 2
        ....
```

Function mbc_mask_write_reg()

This function writes the bits of a specified HOLDING REGISTER to a modbus server.

Syntax	**i32 mbc_mask_write_reg (i32 mbc, i32 offsHRStart, i32 endian, i32 bitMsk, i32 bitVal)**
mbc	Handle value returned by mbc_init()
offsHRStart	Initial index of the HOLDING REGISTER
endian	Type of endian to be used:

Value	Meaning
0 [EM_B_16]	Big Endian in the single word. (default)

Syntax	**i32 mbc_mask_write_reg (i32 mbc, i32 offsHRStart, i32 endian, i32 bitMsk, i32 bitVal)**
	<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>1 [EM_L_16]</td><td>Little Endian in the single word</td></tr><tr><td>2 [EM_B_32]</td><td>Big Endian in 32 bits</td></tr><tr><td>3 [EM_L_32]</td><td>Little Endian in 32 bits</td></tr><tr><td>4 [EM_MB_32]</td><td>Middle Big Endian in 32 bits</td></tr><tr><td>5 [EM_ML_32]</td><td>Middle little Endian in 32 bits</td></tr><tr><td>6 [EM_B_64]</td><td>Big Endian in 64 bit</td></tr><tr><td>7 [EM_L_64]</td><td>Little Endian in 64 bit</td></tr><tr><td>8 [EM_MB_64]</td><td>Middle Big Endian in 64 bit</td></tr><tr><td>9 [EM_ML_64]</td><td>Middle little Endian in 64 bit</td></tr></table>
bitMsk	Mask of the bits that have to be written. **Note**: The mask must have the affected bits in 1 (R3-style, as opposed to what modbus wants instead). `result = (result r_and r_not bitMsk) r_or (bitVal r_and bitMsk)`
bitVal	Mask containing the state of the affected bits
Result	Returns: <table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0 [MBC_OK]</td><td>Command executed</td></tr><tr><td>-1 [MBC_INVAL]</td><td>Invalid arguments</td></tr><tr><td>-2 [MBC_BUSY]</td><td>Resource occupied</td></tr><tr><td>-3 [MBC_PERM]</td><td>Permission denied</td></tr><tr><td>-4 [MBC_CONN]</td><td>Connection failed</td></tr><tr><td>-5 [MBC_NOBUFS]</td><td>Buffer exhausted</td></tr><tr><td>-6 [MBC_NOMEM]</td><td>Insufficient memory</td></tr><tr><td>-7 [MBC_TX_ERR]</td><td>Transmission error</td></tr></table>

© 2025 Robox SpA

Syntax	**i32 mbc_mask_write_reg (i32 mbc, i32 offsHRStart, i32 endian, i32 bitMsk, i32 bitVal)**

Value	Meaning
-8 [MBC_RX_ERR]	Receive error
-9 [MBC_TIMEOUT]	Blocking command timeout
-10 [MBC_EXC_CODE_1]	Response with exception 1 (function code not supported by the server)
-11 [MBC_EXC_CODE_2]	Response with exception 2 (invalid address)
-12 [MBC_EXC_CODE_3]	Response with exception 3 (invalid quantity or value)
-13 [MBC_EXC_CODE_4]	Response with exception 4 (operation failed)
-14 [MBC_WRONG_ANSWER]	Invalid response
-15 [MBC_INVAL_MBC]	Invalid *mbc* id

Validity	Task
Note	The function is blocking. It exits only when the operation has been completed

Function mbc_mask_write_reg_s()

This function writes the bits of a specified HOLDING REGISTER to a modbus server via a support structure in pass-through mode.

Syntax	**i32 mbc_mask_write_reg_s (i32 mbc, stru_modbus stru, i32 bitMsk, i32 bitVal)**
mbc	Handle value returned by mbc_init 366()
stru	Structure of type stru_modbus 847 containing the parameters
bitMsk	Mask of the bits that have to be written. **Note**: The mask must have the affected bits in 1 (R3-style, as opposed to what modbus wants instead).

Syntax	**i32 mbc_mask_write_reg_s (i32 mbc, stru_modbus stru, i32 bitMsk, i32 bitVal)**
	`result = (result r_and r_not bitMsk) r_or (bitVal r_and bitMsk)`
bitVal	Mask containing the state of the affected bits
Result	Returns:

Value	Meaning
1 [MBC_WAIT]	Command waiting for response
0 [MBC_OK]	Command executed
-1 [MBC_INVAL]	Invalid arguments
-2 [MBC_BUSY]	Resource occupied
-3 [MBC_PERM]	Permission denied
-4 [MBC_CONN]	Connection failed
-5 [MBC_NOBUFS]	Buffer exhausted
-6 [MBC_NOMEM]	Insufficient memory
-7 [MBC_TX_ERR]	Transmission error
-8 [MBC_RX_ERR]	Receive error
-9 [MBC_TIMEOUT]	Blocking command timeout
-10 [MBC_EXC_CODE_1]	Response with exception 1 (function code not supported by server)
-11 [MBC_EXC_CODE_2]	Response with exception 2 (invalid address)
-12 [MBC_EXC_CODE_3]	Response with exception 3 (invalid quantity or value)
-13 [MBC_EXC_CODE_4]	Response with exception 4 (operation failed)
-14 [MBC_WRONG_ANSWER]	Invalid response
-15 [MBC_INVAL_MBC]	Invalid *mbc* id

© 2025 Robox SpA

Syntax	**i32 mbc_mask_write_reg_s (i32 mbc, stru_modbus stru, i32 bitMsk, i32 bitVal)**
Validity	Task
Note	The function is pass-through. As long as it returns MBC_WAIT, it must be continuously called within the task

Function mbc_read_coils()

Function for reading COILS on a modbus server.

Syntax	**i32 mbc_read_coils (i32 mbc, i32 offsCoilsStart, i32 nCoil, dest)**
mbc	Handle value returned by mbc_init 366()
offsCoilsStart	Initial index of the COIL
nCoil	Number of COILS to be read from *offsCoilsStart*. **NOTE**: The maximum number of COILS that can be read is 2000.
dest	Area where to put the read data. Can be i16/u16/i32 or array of i16/u16/i32
Result	Returns:

Value	Meaning
0 [MBC_OK]	Command executed
-1 [MBC_INVAL]	Invalid arguments
-2 [MBC_BUSY]	Resource occupied
-3 [MBC_PERM]	Permission denied
-4 [MBC_CONN]	Connection failed
-5 [MBC_NOBUFS]	Buffer exhausted
-6 [MBC_NOMEM]	Insufficient memory
-7 [MBC_TX_ERR]	Transmission error
-8 [MBC_RX_ERR]	Receive error
-9 [MBC_TIMEOUT]	Blocking command timeout
-10 [MBC_EXC_CODE_1]	Response with exception 1 (function code not supported by server)

© 2025 Robox SpA

Syntax	**i32 mbc_read_coils (i32 mbc, i32 offsCoilsStart, i32 nCoil, dest)**
	<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>-11 [MBC_EXC_CODE_2]</td><td>Response with exception 2 (invalid address)</td></tr><tr><td>-12 [MBC_EXC_CODE_3]</td><td>Response with exception 3 (invalid quantity or value)</td></tr><tr><td>-13 [MBC_EXC_CODE_4]</td><td>Response with exception 4 (operation failed)</td></tr><tr><td>-14 [MBC_WRONG_ANSWER]</td><td>Invalid response</td></tr><tr><td>-15 [MBC_INVAL_MBC]</td><td>Invalid *mbc* id</td></tr></table>
Validity	Task
Note	The function is blocking. Exits only when the task is completed

Function mbc_read_coils_s()

Function for reading COILS on a modbus server via support structure in pass-through mode.

Syntax	**i32 mbc_read_coils_s (i32 mbc, stru_modbus stru, dest)**
mbc	Handle value returned by mbc_init 366()
stru	Structure of type stru_modbus 847 containing the parameters
dest	Area where to put the read data. Can be i16/u16/i32 or array of i16/u16/i32
Result	Returns: <table><tr><th>Value</th><th>Meaning</th></tr><tr><td>1 [MBC_WAIT]</td><td>Command waiting for response</td></tr><tr><td>0 [MBC_OK]</td><td>Command executed</td></tr><tr><td>-1 [MBC_INVAL]</td><td>Invalid arguments</td></tr><tr><td>-2 [MBC_BUSY]</td><td>Resource occupied</td></tr></table>

© 2025 Robox SpA

Syntax	**i32 mbc_read_coils_s (i32 mbc, stru_modbus stru, dest)**

Value	Meaning
-3 [MBC_PERM]	Permission denied
-4 [MBC_CONN]	Connection failed
-5 [MBC_NOBUFS]	Buffer exhausted
-6 [MBC_NOMEM]	Insufficient memory
-7 [MBC_TX_ERR]	Transmission error
-8 [MBC_RX_ERR]	Receive error
-9 [MBC_TIMEOUT]	Blocking command timeout
-10 [MBC_EXC_CODE_1]	Response with exception 1 (function code not supported by server)
-11 [MBC_EXC_CODE_2]	Response with exception 2 (invalid address)
-12 [MBC_EXC_CODE_3]	Response with exception 3 (invalid quantity or value)
-13 [MBC_EXC_CODE_4]	Response with exception 4 (operation failed)
-14 [MBC_WRONG_ANSWER]	Invalid response
-15 [MBC_INVAL_MBC]	Invalid *mbc* id

Validity	Task
Note	The function is pass-through. As long as it returns MBC_WAIT, it must be continuously called within the task

Function mbc_read_discr_inp()

Function for reading DISCRETE INPUTS on modbus server.

Syntax	**i32 mbc_read_discr_inp (i32 mbc, i32 offsDIStart, i32 nDI, dest)**
mbc	Handle value returned by mbc_init 366()

Syntax	**i32 mbc_read_discr_inp (i32 mbc, i32 offsDIStart, i32 nDI, dest)**
offsDIStart	Initial index of the DISCRETE INPUT
nDI	Number of DISCRETE INPUTS to be read from offsDIStart
dest	Area where to put the read data. Can be i16/u16/i32 or array of i16/u16/i32
Result	Returns:

Value	Meaning
0 [MBC_OK]	Command executed
-1 [MBC_INVAL]	Invalid arguments
-2 [MBC_BUSY]	Resource occupied
-3 [MBC_PERM]	Permission denied
-4 [MBC_CONN]	Connection failed
-5 [MBC_NOBUFS]	Buffer exhausted
-6 [MBC_NOMEM]	Insufficient memory
-7 [MBC_TX_ERR]	Transmission error
-8 [MBC_RX_ERR]	Receive error
-9 [MBC_TIMEOUT]	Blocking command timeout
-10 [MBC_EXC_CODE_1]	Response with exception 1 (function code not supported by server)
-11 [MBC_EXC_CODE_2]	Response with exception 2 (invalid address)
-12 [MBC_EXC_CODE_3]	Response with exception 3 (invalid quantity or value)
-13 [MBC_EXC_CODE_4]	Response with exception 4 (operation failed)
-14 [MBC_WRONG_ANSWER]	Invalid response
-15 [MBC_INVAL_MBC]	Invalid *mbc* id

© 2025 Robox SpA

Syntax	**i32 mbc_read_discr_inp (i32 mbc, i32 offsDIStart, i32 nDI, dest)**
Validity	Task
Note	The function is blocking. Exits only when the task is completed

Function mbc_read_discr_inp_s()

Function for reading DISCRETE INPUTS on specified modbus servers via a pass-through mode support structure.

Syntax	**i32 mbc_read_discr_inp_s (i32 mbc, modbus stru, dest)**
mbc	Handle value returned by mbc_init 366()
stru	Structure type stru_modbus 847 containing the parameters
dest	Area where to put read data. Can be i16/u16/i32 or array of i16/u16/i32
Result	Returns:

Value	Meaning
1 [MBC_WAIT]	Command waiting for response
0 [MBC_OK]	Command executed
-1 [MBC_INVAL]	Invalid arguments
-2 [MBC_BUSY]	Resource occupied
-3 [MBC_PERM]	Permission denied
-4 [MBC_CONN]	Connection failed
-5 [MBC_NOBUFS]	Buffer exhausted
-6 [MBC_NOMEM]	Insufficient memory
-7 [MBC_TX_ERR]	Transmission error
-8 [MBC_RX_ERR]	Receive error
-9 [MBC_TIMEOUT]	Blocking command timeout
-10 [MBC_EXC_CODE_1]	Response with exception 1 (function code not supported by server)

© 2025 Robox SpA

Syntax	**i32 mbc_read_discr_inp_s (i32 mbc, modbus stru, dest)**
	Value / Meaning
	-11 [MBC_EXC_CODE_2] — Response with exception 2 (invalid address)
	-12 [MBC_EXC_CODE_3] — Response with exception 3 (invalid quantity or value)
	-13 [MBC_EXC_CODE_4] — Response with exception 4 (operation failed)
	-14 [MBC_WRONG_ANSWER] — Invalid response
	-15 [MBC_INVAL_MBC] — Invalid *mbc* id
Validity	Task
Note	The function is pass-through. As long as it returns MBC_WAIT, it must be continuously called within the task

Function mbc_read_hold_reg()

Function for reading HOLDING REGISTER specified on modbus server.

Syntax	**i32 mbc_read_hold_reg (i32 mbc, i32 offsHRStart, i32 nHR, i32 endian, dest)**
mbc	Handle value returned by mbc_init 366()
offsHRStart	Initial index of the HOLDING REGISTER
nHR	Number of HOLDING REGISTER to read **NOTE**: The maximum number of HOLDING REGISTERs that can be read is 125.
endian	Type of endian to be used:
	Value / Meaning
	0 [EM_B_16] — Big Endian nella singola word (default)
	1 [EM_L_16] — Little Endian nella singola word

© 2025 Robox SpA

Syntax	**i32 mbc_read_hold_reg (i32 mbc, i32 offsHRStart, i32 nHR, i32 endian, dest)**

Value	Meaning
2 [EM_B_32]	Big Endian in 32 bit
3 [EM_L_32]	Little Endian in 32 bit
4 [EM_MB_32]	Middle Big Endian in 32 bit
5 [EM_ML_32]	Middle little Endian in 32 bit
6 [EM_B_64]	Big Endian in 64 bit
7 [EM_L_64]	Little Endian in 64 bit
8 [EM_MB_64]	Middle Big Endian in 64 bit
9 [EM_ML_64]	Middle little Endian in 64 bit

dest	Area where to put read data. Can be i16/u16/i32/float/real or array of i16/u16/i32/float/real
Result	Return:

Value	Meaning
0 [MBC_OK]	Command executed
-1 [MBC_INVAL]	Invalid arguments
-2 [MBC_BUSY]	Resource occupied
-3 [MBC_PERM]	Permission denied
-4 [MBC_CONN]	Connection failed
-5 [MBC_NOBUFS]	Buffer exhausted
-6 [MBC_NOMEM]	Insufficient memory
-7 [MBC_TX_ERR]	Transmission error
-8 [MBC_RX_ERR]	Reception error
-9 [MBC_TIMEOUT]	Timeout blocking command
-10 [MBC_EXC_CODE_1]	Response with exception 1 (function code not supported by the server)

© 2025 Robox SpA

Syntax	**i32 mbc_read_hold_reg (i32 mbc, i32 offsHRStart, i32 nHR, i32 endian, dest)**
	Value / Meaning
	-11 [MBC_EXC_CODE_2] — Response with exception 2 (invalid address)
	-12 [MBC_EXC_CODE_3] — Response with exception 3 (invalid quantity or value)
	-13 [MBC_EXC_CODE_4] — Response with exception 4 (operation failed)
	-14 [MBC_WRONG_ANSWER] — Invalid response
	-15 [MBC_INVAL_MBC] — Invalid *mbc* id
Validity	Task
Note	The function is blocking. It exits only when the operation is completed

Function mbc_read_hold_reg_s()

Function for reading HOLDING REGISTERs specified on modbus server via a support structure in pass-through mode.

Syntax	**i32 mbc_read_hold_reg_s (i32 mbc, modbus stru, dest)**
mbc	Handle value returned by mbc_init 366()
stru	Structure of type stru_modbus 847 containing the parameters
dest	Area where to put read data. Can be i16/u16/i32/float/real or array of i16/u16/i32/float/real
Result	Returns:
	Value / Meaning
	1 [MBC_WAIT] — Command waiting for response
	0 [MBC_OK] — Command executed
	-1 [MBC_INVAL] — Invalid arguments

© 2025 Robox SpA

Syntax	**i32 mbc_read_hold_reg_s (i32 mbc, modbus stru, dest)**

Value	Meaning
-2 [MBC_BUSY]	Resource occupied
-3 [MBC_PERM]	Permission denied
-4 [MBC_CONN]	Connection failed
-5 [MBC_NOBUFS]	Buffer exhausted
-6 [MBC_NOMEM]	Insufficient memory
-7 [MBC_TX_ERR]	Transmission error
-8 [MBC_RX_ERR]	Receive error
-9 [MBC_TIMEOUT]	Blocking command timeout
-10 [MBC_EXC_CODE_1]	Response with exception 1 (function code not supported by server)
-11 [MBC_EXC_CODE_2]	Response with exception 2 (invalid address)
-12 [MBC_EXC_CODE_3]	Response with exception 3 (invalid quantity or value)
-13 [MBC_EXC_CODE_4]	Response with exception 4 (operation failed)
-14 [MBC_WRONG_ANSWER]	Invalid response
-15 [MBC_INVAL_MBC]	Invalid *mbc* id

Validity	Task
Note	The function is pass-through. As long as it returns MBC_WAIT, it must be continuously called within the task

© 2025 Robox SpA

Function mbc_read_inp_reg()

Function for reading INPUT REGISTER specified on modbus server.

Syntax	**i32 mbc_read_inp_reg (i32 mbc, i32 offsIRStart, i32 nIR, i32 endian, dest)**
mbc	Handle value returned by mbc_init 366()
offsIRStart	Initial index of the INPUT REGISTER
nIR	Number of INPUT REGISTER to be read
endian	Type of endian to be used:

Value	Meaning
0 [EM_B_16]	Big Endian in the single word. (default)
1 [EM_L_16]	Little Endian in the single word
2 [EM_B_32]	Big Endian in 32 bits
3 [EM_L_32]	Little Endian in 32 bits
4 [EM_MB_32]	Middle Big Endian in 32 bits
5 [EM_ML_32]	Middle little Endian in 32 bits
6 [EM_B_64]	Big Endian in 64 bits
7 [EM_L_64]	Little Endian in 64 bits
8 [EM_MB_64]	Middle Big Endian in 64 bits
9 [EM_ML_64]	Middle little Endian in 64 bits

dest	Area where to put read data. Can be i16/u16/i32/float/real or array of i16/u16/i32/float/real
Result	Returns:

Value	Meaning
0 [MBC_OK]	Command executed
-1 [MBC_INVAL]	Invalid arguments
-2 [MBC_BUSY]	Resource occupied
-3 [MBC_PERM]	Permission denied
-4 [MBC_CONN]	Connection failed

© 2025 Robox SpA

Syntax	**i32 mbc_read_inp_reg (i32 mbc, i32 offsIRStart, i32 nIR, i32 endian, dest)**

Value	Meaning
-5 [MBC_NOBUFS]	Buffer exhausted
-6 [MBC_NOMEM]	Insufficient memory
-7 [MBC_TX_ERR]	Transmission error
-8 [MBC_RX_ERR]	Receive error
-9 [MBC_TIMEOUT]	Blocking command timeout
-10 [MBC_EXC_CODE_1]	Response with exception 1 (function code not supported by server)
-11 [MBC_EXC_CODE_2]	Response with exception 2 (invalid address)
-12 [MBC_EXC_CODE_3]	Response with exception 3 (invalid quantity or value)
-13 [MBC_EXC_CODE_4]	Response with exception 4 (operation failed)
-14 [MBC_WRONG_ANSWER]	Invalid response
-15 [MBC_INVAL_MBC]	Invalid *mbc* id

Validity	Task
Note	The function is blocking. Exits only when the task is completed

Function mbc_read_inp_reg_s()

Function for reading INPUT REGISTER specified INPUT REGISTER on modbus server via a support structure in pass-through mode.

Syntax	**i32 mbc_read_inp_reg_s (i32 mbc, modbus stru, dest)**
mbc	Handle value returned by mbc_init 366()
stru	Structure of type stru_modbus 847 containing the parameters

Syntax	**i32 mbc_read_inp_reg_s (i32 mbc, modbus stru, dest)**
dest	Area where to put read data. Can be i16/u16/i32/float/real or array of i16/u16/i32/float/real
Result	Returns:

Value	Meaning
1 [MBC_WAIT]	Command waiting for response
0 [MBC_OK]	Command executed
-1 [MBC_INVAL]	Invalid arguments
-2 [MBC_BUSY]	Resource occupied
-3 [MBC_PERM]	Permission denied
-4 [MBC_CONN]	Connection failed
-5 [MBC_NOBUFS]	Buffer exhausted
-6 [MBC_NOMEM]	Insufficient memory
-7 [MBC_TX_ERR]	Transmission error
-8 [MBC_RX_ERR]	Receive error
-9 [MBC_TIMEOUT]	Blocking command timeout
-10 [MBC_EXC_CODE_1]	Response with exception 1 (function code not supported by server)
-11 [MBC_EXC_CODE_2]	Response with exception 2 (invalid address)
-12 [MBC_EXC_CODE_3]	Response with exception 3 (invalid quantity or value)
-13 [MBC_EXC_CODE_4]	Response with exception 4 (operation failed)
-14 [MBC_WRONG_ANSWER]	Invalid response
-15 [MBC_INVAL_MBC]	Invalid *mbc* id

Validity	Task

© 2025 Robox SpA

Syntax	**i32 mbc_read_inp_reg_s (i32 mbc, modbus stru, dest)**
Note	The function is pass-through. As long as it returns MBC_WAIT, it must be continuously called within the task

Function mbc_shutdown()

Function used to close the connection with the specified Modbus TCP server.

Syntax	**i32 mbc_shutdown (i32 mbc)**
mbc	Handle Modbus returned by the previous function mbc_init 366()
Result	Returns: Value — Meaning >= 0 — ERROR if OK disconnect operation or one of the reference MBC_const codes 62 -1 — Illegal arguments -4 — Client already disconnected from the server (MBC_CONN)
Validity	Task
System Requirements	From RTE 34.30.0
Note	

Function mbc_write_coils()

This function writes the specified COILS to a modbus server.

Syntax	**i32 mbc_write_coils (i32 mbc, i32 offsCoilStart, i32 nCoil, source)**
mbc	Handle value returned by mbc_init 366()
offsCoilStart	Initial index of the COIL
nCoil	Number of COILS to be written. **NOTE**: The maximum number of COILS that can be written is 1968.
source	Area where to take the data to be written. Can be i16/u16/i32 or array of i16/u16/i32

© 2025 Robox SpA

Syntax	**i32 mbc_write_coils (i32 mbc, i32 offsCoilStart, i32 nCoil, source)**
Result	Returns:

Value	Meaning
0 [MBC_OK]	Command executed
-1 [MBC_INVAL]	Invalid arguments
-2 [MBC_BUSY]	Resource occupied
-3 [MBC_PERM]	Permission denied
-4 [MBC_CONN]	Connection failed
-5 [MBC_NOBUFS]	Buffer exhausted
-6 [MBC_NOMEM]	Insufficient memory
-7 [MBC_TX_ERR]	Transmission error
-8 [MBC_RX_ERR]	Receive error
-9 [MBC_TIMEOUT]	Blocking command timeout
-10 [MBC_EXC_CODE_1]	Response with exception 1 (function code not supported by server)
-11 [MBC_EXC_CODE_2]	Response with exception 2 (invalid address)
-12 [MBC_EXC_CODE_3]	Response with exception 3 (invalid quantity or value)
-13 [MBC_EXC_CODE_4]	Response with exception 4 (operation failed)
-14 [MBC_WRONG_ANSWER]	Invalid response
-15 [MBC_INVAL_MBC]	Invalid *mbc* id

Validity	Task
Note	The function is blocking. Exits only when the task is completed

© 2025 Robox SpA

Function mbc_write_coils_s()

This function writes the specified COILS to a modbus server via a support structure in pass-through mode.

Syntax	**i32 mbc_write_coils_s (i32 mbc, stru_modbus stru, source)**
mbc	Handle value returned by mbc_init 366()
stru	Structure of type stru_modbus 847 containing the parameters
source	Area where to take data to be written. Can be i16/u16/i32/float/real or array of i16/u16/i32/float/real
Result	Returns:

Value	Meaning
1 [MBC_WAIT]	Command waiting for response
0 [MBC_OK]	Command executed
-1 [MBC_INVAL]	Invalid arguments
-2 [MBC_BUSY]	Resource occupied
-3 [MBC_PERM]	Permission denied
-4 [MBC_CONN]	Connection failed
-5 [MBC_NOBUFS]	Buffer exhausted
-6 [MBC_NOMEM]	Insufficient memory
-7 [MBC_TX_ERR]	Transmission error
-8 [MBC_RX_ERR]	Receive error
-9 [MBC_TIMEOUT]	Blocking command timeout
-10 [MBC_EXC_CODE_1]	Response with exception 1 (function code not supported by server)
-11 [MBC_EXC_CODE_2]	Response with exception 2 (invalid address)
-12 [MBC_EXC_CODE_3]	Response with exception 3 (invalid quantity or value)
-13 [MBC_EXC_CODE_4]	Response with exception 4 (operation failed)

© 2025 Robox SpA

Syntax	**i32 mbc_write_coils_s (i32 mbc, stru_modbus stru, source)**
	Value / Meaning -14 [MBC_WRONG_ANSWER] — Invalid response -15 [MBC_INVAL_MBC] — Invalid *mbc* id
Validity	Task
Note	The function is pass-through. As long as it returns MBC_WAIT, it must be continuously called within the task

Function mbc_write_multiple_coils()

This function writes the specified COILS always using the Modbus function code of the corresponding multiple writes FC 15, unlike mbc_write_coils 384() which can also use FC 05 (if single COIL).

Syntax	**i32 mbc_write_multiple_coils(i32 mbc, i32 offsCoilStart, i32 nCoil, source)**
mbc	Handle value returned by mbc_init 366()
offsCoilStart	Initial index of the COIL
nCoil	Number of COILS to be written. **NOTE**: The maximum number of COILS that can be written is 1968.
source	Area where to take the data to be written. Can be i16/u16/i32 or array of i16/u16/i32
Result	Returns: Value / Meaning 0 [MBC_OK] — Command executed -1 [MBC_INVAL] — Invalid arguments -2 [MBC_BUSY] — Resource occupied -3 [MBC_PERM] — Permission denied -4 [MBC_CONN] — Connection failed -5 [MBC_NOBUFS] — Buffer exhausted -6 [MBC_NOMEM] — Insufficient memory

© 2025 Robox SpA

Syntax	**i32 mbc_write_multiple_coils(i32 mbc, i32 offsCoilStart, i32 nCoil, source)**

Value	Meaning
-7 [MBC_TX_ERR]	Transmission error
-8 [MBC_RX_ERR]	Receive error
-9 [MBC_TIMEOUT]	Blocking command timeout
-10 [MBC_EXC_CODE_1]	Response with exception 1 (function code not supported by server)
-11 [MBC_EXC_CODE_2]	Response with exception 2 (invalid address)
-12 [MBC_EXC_CODE_3]	Response with exception 3 (invalid quantity or value)
-13 [MBC_EXC_CODE_4]	Response with exception 4 (operation failed)
-14 [MBC_WRONG_ANSWER]	Invalid response
-15 [MBC_INVAL_MBC]	Invalid *mbc* id

Validity	Task
Note	The function is blocking. Exits only when the task is completed

Function mbc_write_multiple_coils_s()

This function writes the specified COILS to a Modbus server via a support structure in pass-through mode and always using the Modbus function code of the corresponding multiple writes FC 15, unlike mbc_write_coils_s 386() which can also use FC 05.

Syntax	**i32 mbc_write_multiple_coils_s (i32 mbc, stru_modbus stru, source)**
mbc	Handle value returned by mbc_init 366()
stru	Structure of type stru_modbus 847 containing the parameters
source	Area where to take data to be written.

Syntax	**i32 mbc_write_multiple_coils_s (i32 mbc, stru_modbus stru, source)**
	Can be i16/u16/i32/float/real or array of i16/u16/i32/float/real
Result	Returns:

Value	Meaning
1 [MBC_WAIT]	Command waiting for response
0 [MBC_OK]	Command executed
-1 [MBC_INVAL]	Invalid arguments
-2 [MBC_BUSY]	Resource occupied
-3 [MBC_PERM]	Permission denied
-4 [MBC_CONN]	Connection failed
-5 [MBC_NOBUFS]	Buffer exhausted
-6 [MBC_NOMEM]	Insufficient memory
-7 [MBC_TX_ERR]	Transmission error
-8 [MBC_RX_ERR]	Receive error
-9 [MBC_TIMEOUT]	Blocking command timeout
-10 [MBC_EXC_CODE_1]	Response with exception 1 (function code not supported by server)
-11 [MBC_EXC_CODE_2]	Response with exception 2 (invalid address)
-12 [MBC_EXC_CODE_3]	Response with exception 3 (invalid quantity or value)
-13 [MBC_EXC_CODE_4]	Response with exception 4 (operation failed)
-14 [MBC_WRONG_ANSWER]	Invalid response
-15 [MBC_INVAL_MBC]	Invalid *mbc* id

Validity	Task

© 2025 Robox SpA

Syntax	**i32 mbc_write_multiple_coils_s (i32 mbc, stru_modbus stru, source)**
Note	The function is pass-through. As long as it returns MBC_WAIT, it must be continuously called within the task

Function mbc_write_multiple_regs()

This function writes the specified HOLDING REGISTERS to a Modbus server using the Modbus function code of the corresponding multiple writes FC 16, unlike mbc_write_reg 393() which can also use FC 06.

Syntax	**i32 mbc_write_multiple_regs (i32 mbc, i32 offsHRStart, i32 nHR, i32 endian, source)**
mbc	Handle value returned by mbc_init 366()
offsHRStart	Initial index of the HOLDING REGISTER
nHR	Number of HOLDING REGISTERs to be written. **NOTE**: The maximum number of HOLDING REGISTERs that can be written is 123.
endian	Type of endian to be used:

Value	Meaning
0 [EM_B_16]	Big Endian in the single word. (default)
1 [EM_L_16]	Little Endian in the single word
2 [EM_B_32]	Big Endian in 32 bits
3 [EM_L_32]	Little Endian in 32 bits
4 [EM_MB_32]	Middle Big Endian in 32 bits
5 [EM_ML_32]	Middle little Endian in 32 bits
6 [EM_B_64]	Big Endian in 64 bits
7 [EM_L_64]	Little Endian in 64 bits
8 [EM_MB_64]	Middle Big Endian in 64 bits
9 [EM_ML_64]	Middle little Endian in 64 bits

source	Area where to take data to write. Can be i16/u16/i32/float/real or array of i16/u16/i32/float/real

© 2025 Robox SpA

Syntax	**i32 mbc_write_multiple_regs (i32 mbc, i32 offsHRStart, i32 nHR, i32 endian, source)**
Result	Returns:

Value	Meaning
0 [MBC_OK]	Command executed
-1 [MBC_INVAL]	Invalid arguments
-2 [MBC_BUSY]	Resource occupied
-3 [MBC_PERM]	Permission denied
-4 [MBC_CONN]	Connection failed
-5 [MBC_NOBUFS]	Buffer exhausted
-6 [MBC_NOMEM]	Insufficient memory
-7 [MBC_TX_ERR]	Transmission error
-8 [MBC_RX_ERR]	Receive error
-9 [MBC_TIMEOUT]	Blocking command timeout
-10 [MBC_EXC_CODE_1]	Response with exception 1 (function code not supported by server)
-11 [MBC_EXC_CODE_2]	Response with exception 2 (invalid address)
-12 [MBC_EXC_CODE_3]	Response with exception 3 (invalid quantity or value)
-13 [MBC_EXC_CODE_4]	Response with exception 4 (operation failed)
-14 [MBC_WRONG_ANSWER]	Invalid response
-15 [MBC_INVAL_MBC]	Invalid *mbc* id

Validity	Task
Note	The function is blocking. Exits only when the task is completed

© 2025 Robox SpA

Function mbc_write_multiple_regs_s()

This function writes the specified HOLDING REGISTERS to a Modbus server via a support structure in pass-through mode and using the Modbus function code of the corresponding multiple writes FC 16, unlike mbc_write_reg_s 395() which can also use FC 06.

Syntax	**i32 mbc_write_multiple_regs_s (i32 mbc, stru_modbus stru, source)**
mbc	Handle value returned by mbc_init 366()
stru	Structure of type stru_modbus 847 containing the parameters
source	Area where to take data to be written. Can be i16/u16/i32/float/real or array of i16/u16/i32/float/real
Result	Returns:

Value	Meaning
1 [MBC_WAIT]	Command waiting for response
0 [MBC_OK]	Command executed
-1 [MBC_INVAL]	Invalid arguments
-2 [MBC_BUSY]	Resource occupied
-3 [MBC_PERM]	Permission denied
-4 [MBC_CONN]	Connection failed
-5 [MBC_NOBUFS]	Buffer exhausted
-6 [MBC_NOMEM]	Insufficient memory
-7 [MBC_TX_ERR]	Transmission error
-8 [MBC_RX_ERR]	Receive error
-9 [MBC_TIMEOUT]	Blocking command timeout
-10 [MBC_EXC_CODE_1]	Response with exception 1 (function code not supported by server)
-11 [MBC_EXC_CODE_2]	Response with exception 2 (invalid address)
-12 [MBC_EXC_CODE_3]	Response with exception 3 (invalid quantity or value)
-13 [MBC_EXC_CODE_4]	Response with exception 4

© 2025 Robox SpA

Syntax	**i32 mbc_write_multiple_regs_s (i32 mbc, stru_modbus stru, source)**
	Value / Meaning: (operation failed) -14 [MBC_WRONG_ANSWER] — Invalid response -15 [MBC_INVAL_MBC] — Invalid *mbc* id
Validity	Task
Note	The function is pass-through. As long as it returns MBC_WAIT, it must be continuously called within the task

Function mbc_write_reg()

This function writes the specified HOLDING REGISTERS to a modbus server.

Syntax	**i32 mbc_write_reg (i32 mbc, i32 offsHRStart, i32 nHR, i32 endian, source)**
mbc	Handle value returned by mbc_init 366()
offsHRStart	Initial index of the HOLDING REGISTER
nHR	Number of HOLDING REGISTERs to be written. **NOTE**: The maximum number of HOLDING REGISTERs that can be written is 123.
endian	Type of endian to be used: Value / Meaning: 0 [EM_B_16] — Big Endian in the single word. (default) 1 [EM_L_16] — Little Endian in the single word 2 [EM_B_32] — Big Endian in 32 bits 3 [EM_L_32] — Little Endian in 32 bits 4 [EM_MB_32] — Middle Big Endian in 32 bits 5 [EM_ML_32] — Middle little Endian in 32 bits 6 [EM_B_64] — Big Endian in 64 bits

© 2025 Robox SpA

Syntax	**i32 mbc_write_reg (i32 mbc, i32 offsHRStart, i32 nHR, i32 endian, source)**

Value	Meaning
7 [EM_L_64]	Little Endian in 64 bits
8 [EM_MB_64]	Middle Big Endian in 64 bits
9 [EM_ML_64]	Middle little Endian in 64 bits

source	Area where to take data to write. Can be i16/u16/i32/float/real or array of i16/u16/i32/float/real
Result	Returns:

Value	Meaning
0 [MBC_OK]	Command executed
-1 [MBC_INVAL]	Invalid arguments
-2 [MBC_BUSY]	Resource occupied
-3 [MBC_PERM]	Permission denied
-4 [MBC_CONN]	Connection failed
-5 [MBC_NOBUFS]	Buffer exhausted
-6 [MBC_NOMEM]	Insufficient memory
-7 [MBC_TX_ERR]	Transmission error
-8 [MBC_RX_ERR]	Receive error
-9 [MBC_TIMEOUT]	Blocking command timeout
-10 [MBC_EXC_CODE_1]	Response with exception 1 (function code not supported by server)
-11 [MBC_EXC_CODE_2]	Response with exception 2 (invalid address)
-12 [MBC_EXC_CODE_3]	Response with exception 3 (invalid quantity or value)
-13 [MBC_EXC_CODE_4]	Response with exception 4 (operation failed)

© 2025 Robox SpA

Syntax	**i32 mbc_write_reg (i32 mbc, i32 offsHRStart, i32 nHR, i32 endian, source)**
	Value / Meaning -14 [MBC_WRONG_ANSWER] — Invalid response -15 [MBC_INVAL_MBC] — Invalid *mbc* id
Validity	Task
Note	The function is blocking. Exits only when the task is completed

Function mbc_write_reg_s()

This function writes the specified HOLDING REGISTERS to a modbus server via a support structure in pass-through mode.

Syntax	**i32 mbc_write_reg_s (i32 mbc, stru_modbus stru, source)**
mbc	Handle value returned by mbc_init 366()
stru	Structure of type stru_modbus 847 containing the parameters
source	Area where to take data to be written. Can be i16/u16/i32/float/real or array of i16/u16/i32/float/real
Result	Returns: Value / Meaning 1 [MBC_WAIT] — Command waiting for response 0 [MBC_OK] — Command executed -1 [MBC_INVAL] — Invalid arguments -2 [MBC_BUSY] — Resource occupied -3 [MBC_PERM] — Permission denied -4 [MBC_CONN] — Connection failed -5 [MBC_NOBUFS] — Buffer exhausted -6 [MBC_NOMEM] — Insufficient memory -7 [MBC_TX_ERR] — Transmission error

© 2025 Robox SpA

Syntax	**i32 mbc_write_reg_s (i32 mbc, stru_modbus stru, source)**

Value	Meaning
-8 [MBC_RX_ERR]	Receive error
-9 [MBC_TIMEOUT]	Blocking command timeout
-10 [MBC_EXC_CODE_1]	Response with exception 1 (function code not supported by server)
-11 [MBC_EXC_CODE_2]	Response with exception 2 (invalid address)
-12 [MBC_EXC_CODE_3]	Response with exception 3 (invalid quantity or value)
-13 [MBC_EXC_CODE_4]	Response with exception 4 (operation failed)
-14 [MBC_WRONG_ANSWER]	Invalid response
-15 [MBC_INVAL_MBC]	Invalid *mbc* id

Validity	Task
Note	The function is pass-through. As long as it returns MBC_WAIT, it must be continuously called within the task

Function mbcu_mask_write_reg()

This function writes the bits of a HOLDING REGISTER to a modbus server by specifying the unit ID.

Syntax	**i32 mbcu_mask_write_reg (i32 mbc, i32 unitID, i32 offsHRStart, i32 endian, i32 bitMsk, i32 bitVal)**
mbc	Handle value returned by mbc_init 366()
unitID	Value of the unit ID to be used
offsHRStart	Initial index of the HOLDING REGISTER
endian	Type of endian to be used:

Syntax	**i32 mbcu_mask_write_reg (i32 mbc, i32 unitID, i32 offsHRStart, i32 endian, i32 bitMsk, i32 bitVal)**
	<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0 [EM_B_16]</td><td>Big Endian in the single word. (default)</td></tr><tr><td>1 [EM_L_16]</td><td>Little Endian in the single word</td></tr><tr><td>2 [EM_B_32]</td><td>Big Endian in 32 bits</td></tr><tr><td>3 [EM_L_32]</td><td>Little Endian in 32 bits</td></tr><tr><td>4 [EM_MB_32]</td><td>Middle Big Endian in 32 bits</td></tr><tr><td>5 [EM_ML_32]</td><td>Middle little Endian in 32 bits</td></tr><tr><td>6 [EM_B_64]</td><td>Big Endian in 64 bits</td></tr><tr><td>7 [EM_L_64]</td><td>Little Endian in 64 bits</td></tr><tr><td>8 [EM_MB_64]</td><td>Middle Big Endian in 64 bits</td></tr><tr><td>9 [EM_ML_64]</td><td>Middle little Endian in 64 bits</td></tr></table>
bitMsk	Mask of the bits that are to be written. **Note**: The mask must have the affected bits in 1 (R3-style, as opposed to what modbus wants instead). `result = (result r_and r_not bitMsk) r_or (bitVal r_and bitMsk)`
bitVal	Mask containing the state of the affected bits
Result	Returns: <table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0 [MBC_OK]</td><td>Command executed</td></tr><tr><td>-1 [MBC_INVAL]</td><td>Invalid arguments</td></tr><tr><td>-2 [MBC_BUSY]</td><td>Resource occupied</td></tr><tr><td>-3 [MBC_PERM]</td><td>Permission denied</td></tr><tr><td>-4 [MBC_CONN]</td><td>Connection failed</td></tr><tr><td>-5 [MBC_NOBUFS]</td><td>Buffer exhausted</td></tr><tr><td>-6 [MBC_NOMEM]</td><td>Insufficient memory</td></tr></table>

© 2025 Robox SpA

Syntax	**i32 mbcu_mask_write_reg (i32 mbc, i32 unitID, i32 offsHRStart, i32 endian, i32 bitMsk, i32 bitVal)**

Value	Meaning
-7 [MBC_TX_ERR]	Transmission error
-8 [MBC_RX_ERR]	Receive error
-9 [MBC_TIMEOUT]	Blocking command timeout
-10 [MBC_EXC_CODE_1]	Response with exception 1 (function code not supported by server)
-11 [MBC_EXC_CODE_2]	Response with exception 2 (invalid address)
-12 [MBC_EXC_CODE_3]	Response with exception 3 (invalid quantity or value)
-13 [MBC_EXC_CODE_4]	Response with exception 4 (operation failed)
-14 [MBC_WRONG_ANSWER]	Invalid response
-15 [MBC_INVAL_MBC]	Invalid *mbc* id

Validity	Task
Note	The function is blocking. Exits only when the task is completed

Function mbcu_read_coils()

Function for reading COILS on a modbus server by specifying the unit ID.

Syntax	**i32 mbcu_read_coils (i32 mbc, i32 unitID, i32 offsCoilsStart, i32 nCoil, dest)**
mbc	Handle value returned by mbc_init 366()
unitID	Value of the unit ID to be used
offsCoilsStart	Initial index of the COIL
nCoil	Number of COILS to be read from *offsCoilsStart*.

© 2025 Robox SpA

Syntax	**i32 mbcu_read_coils (i32 mbc, i32 unitID, i32 offsCoilsStart, i32 nCoil, dest)**
	NOTE: The maximum number of COILS that can be read is 2000.
dest	Area where to put the read data. Can be i16/u16/i32 or array of i16/u16/i32
Result	Returns:

Value	Meaning
0 [MBC_OK]	Command executed
-1 [MBC_INVAL]	Invalid arguments
-2 [MBC_BUSY]	Resource occupied
-3 [MBC_PERM]	Permission denied
-4 [MBC_CONN]	Connection failed
-5 [MBC_NOBUFS]	Buffer exhausted
-6 [MBC_NOMEM]	Insufficient memory
-7 [MBC_TX_ERR]	Transmission error
-8 [MBC_RX_ERR]	Receive error
-9 [MBC_TIMEOUT]	Blocking command timeout
-10 [MBC_EXC_CODE_1]	Response with exception 1 (function code not supported by server)
-11 [MBC_EXC_CODE_2]	Response with exception 2 (invalid address)
-12 [MBC_EXC_CODE_3]	Response with exception 3 (invalid quantity or value)
-13 [MBC_EXC_CODE_4]	Response with exception 4 (operation failed)
-14 [MBC_WRONG_ANSWER]	Invalid response
-15 [MBC_INVAL_MBC]	Invalid *mbc* id

Validity	Task

© 2025 Robox SpA

Syntax	**i32 mbcu_read_coils (i32 mbc, i32 unitID, i32 offsCoilsStart, i32 nCoil, dest)**
Note	The function is blocking. Exits only when the task is completed

Function mbcu_read_discr_inp()

Function for reading DISCRETE INPUTS on modbus server by specifying unit ID.

Syntax	**i32 mbcu_read_discr_inp (i32 mbc, i32 unitID, i32 offsDIStart, i32 nDI, dest)**
mbc	Handle value returned by mbc_init 366()
unitID	Value of the unit ID to be used
offsDIStart	Initial index of the DISCRETE INPUT
nDI	Number of DISCRETE INPUTS to be read from offsDIStart
dest	Area where to put the read data. Can be i16/u16/i32 or array of i16/u16/i32
Result	Returns:

Value	Meaning
0 [MBC_OK]	Command executed
-1 [MBC_INVAL]	Invalid arguments
-2 [MBC_BUSY]	Resource occupied
-3 [MBC_PERM]	Permission denied
-4 [MBC_CONN]	Connection failed
-5 [MBC_NOBUFS]	Buffer exhausted
-6 [MBC_NOMEM]	Insufficient memory
-7 [MBC_TX_ERR]	Transmission error
-8 [MBC_RX_ERR]	Receive error
-9 [MBC_TIMEOUT]	Blocking command timeout
-10 [MBC_EXC_CODE_1]	Response with exception 1 (function code not supported by server)
-11 [MBC_EXC_CODE_2]	Response with exception 2 (invalid address)

© 2025 Robox SpA

Syntax	**i32 mbcu_read_discr_inp (i32 mbc, i32 unitID, i32 offsDIStart, i32 nDI, dest)**
	Value / Meaning: -12 [MBC_EXC_CODE_3] — Response with exception 3 (invalid quantity or value) -13 [MBC_EXC_CODE_4] — Response with exception 4 (operation failed) -14 [MBC_WRONG_ANSWER] — Invalid response -15 [MBC_INVAL_MBC] — Invalid *mbc* id
Validity	Task
Note	The function is blocking. Exits only when the task is completed

Function mbcu_read_hold_reg()

Function for reading HOLDING REGISTER on modbus server by specifying unit ID.

Syntax	**i32 mbcu_read_hold_reg (i32 mbc, i32 unitID, i32 offsHRStart, i32 nHR, i32 endian, dest)**
mbc	Handle value returned by mbc_init 366()
unitID	Value of the unit ID to be used
offsHRStart	Initial index of the HOLDING REGISTER
nHR	Number of HOLDING REGISTERs to be read. **NOTE**: The maximum number of HOLDING REGISTERs that can be read is 125.
endian	Type of endian to be used: Value / Meaning: 0 [EM_B_16] — Big Endian in the single word. (default) 1 [EM_L_16] — Little Endian in the single word 2 [EM_B_32] — Big Endian in 32 bits 3 [EM_L_32] — Little Endian in 32 bits

© 2025 Robox SpA

Syntax	**i32 mbcu_read_hold_reg (i32 mbc, i32 unitID, i32 offsHRStart, i32 nHR, i32 endian, dest)**

Value	Meaning
4 [EM_MB_32]	Middle Big Endian in 32 bits
5 [EM_ML_32]	Middle little Endian in 32 bits
6 [EM_B_64]	Big Endian in 64 bits
7 [EM_L_64]	Little Endian in 64 bits
8 [EM_MB_64]	Middle Big Endian in 64 bits
9 [EM_ML_64]	Middle little Endian in 64 bits

dest	Area where to put read data. Can be i16/u16/i32/float/real or array of i16/u16/i32/float/real
Result	Returns:

Value	Meaning
0 [MBC_OK]	Command executed
-1 [MBC_INVAL]	Invalid arguments
-2 [MBC_BUSY]	Resource occupied
-3 [MBC_PERM]	Permission denied
-4 [MBC_CONN]	Connection failed
-5 [MBC_NOBUFS]	Buffer exhausted
-6 [MBC_NOMEM]	Insufficient memory
-7 [MBC_TX_ERR]	Transmission error
-8 [MBC_RX_ERR]	Receive error
-9 [MBC_TIMEOUT]	Blocking command timeout
-10 [MBC_EXC_CODE_1]	Response with exception 1 (function code not supported by server)
-11 [MBC_EXC_CODE_2]	Response with exception 2 (invalid address)

© 2025 Robox SpA

Syntax	**i32 mbcu_read_hold_reg (i32 mbc, i32 unitID, i32 offsHRStart, i32 nHR, i32 endian, dest)**
	Value / Meaning
	-12 [MBC_EXC_CODE_3] — Response with exception 3 (invalid quantity or value)
	-13 [MBC_EXC_CODE_4] — Response with exception 4 (operation failed)
	-14 [MBC_WRONG_ANSWER] — Invalid response
	-15 [MBC_INVAL_MBC] — Invalid *mbc* id
Validity	Task
Note	The function is blocking. Exits only when the task is completed

Function mbcu_read_inp_reg()

Function for reading INPUT REGISTER on modbus server by specifying unit ID.

Syntax	**i32 mbcu_read_inp_reg (i32 mbc, i32 unitID, i32 offsIRStart, i32 nIR, i32 endian, dest)**
mbc	Handle value returned by mbc_init 366()
unitID	Value of the unit ID to be used
offsIRStart	Initial index of the INPUT REGISTER
nIR	Number of INPUT REGISTER to be read
endian	Type of endian to be used:

Value	Meaning
0 [EM_B_16]	Big Endian in the single word. (default)
1 [EM_L_16]	Little Endian in the single word
2 [EM_B_32]	Big Endian in 32 bits
3 [EM_L_32]	Little Endian in 32 bits
4 [EM_MB_32]	Middle Big Endian in 32 bits

© 2025 Robox SpA

Syntax	**i32 mbcu_read_inp_reg (i32 mbc, i32 unitID, i32 offsIRStart, i32 nIR, i32 endian, dest)**

Value	Meaning
5 [EM_ML_32]	Middle little Endian in 32 bits
6 [EM_B_64]	Big Endian in 64 bits
7 [EM_L_64]	Little Endian in 64 bits
8 [EM_MB_64]	Middle Big Endian in 64 bits
9 [EM_ML_64]	Middle little Endian in 64 bits

dest	Area where to put read data. Can be i16/u16/i32/float/real or array of i16/u16/i32/float/real
Result	Returns:

Value	Meaning
0 [MBC_OK]	Command executed
-1 [MBC_INVAL]	Invalid arguments
-2 [MBC_BUSY]	Resource occupied
-3 [MBC_PERM]	Permission denied
-4 [MBC_CONN]	Connection failed
-5 [MBC_NOBUFS]	Buffer exhausted
-6 [MBC_NOMEM]	Insufficient memory
-7 [MBC_TX_ERR]	Transmission error
-8 [MBC_RX_ERR]	Receive error
-9 [MBC_TIMEOUT]	Blocking command timeout
-10 [MBC_EXC_CODE_1]	Response with exception 1 (function code not supported by server)
-11 [MBC_EXC_CODE_2]	Response with exception 2 (invalid address)
-12 [MBC_EXC_CODE_3]	Response with exception 3 (invalid

© 2025 Robox SpA

Syntax	**i32 mbcu_read_inp_reg (i32 mbc, i32 unitID, i32 offsIRStart, i32 nIR, i32 endian, dest)**
	Value / Meaning
	quantity or value)
	-13 [MBC_EXC_CODE_4] — Response with exception 4 (operation failed)
	-14 [MBC_WRONG_ANSWER] — Invalid response
	-15 [MBC_INVAL_MBC] — Invalid *mbc* id
Validity	Task
Note	The function is blocking. Exits only when the task is completed

Function mbcu_write_coils()

This function writes COILS to a modbus server by specifying the unit ID.

Syntax	**i32 mbcu_write_coils (i32 mbc, i32 unitID, i32 offsCoilStart, i32 nCoil, source)**
mbc	Handle value returned by mbc_init 366()
unitID	Value of the unit ID to be used
offsCoilStart	Initial index of the COIL
nCoil	Number of COILS to be written. **NOTE**: The maximum number of COILS that can be written is 1968.
source	Area where to take the data to be written. Can be i16/u16/i32 or array of i16/u16/i32
Result	Returns:

Value	Meaning
0 [MBC_OK]	Command executed
-1 [MBC_INVAL]	Invalid arguments
-2 [MBC_BUSY]	Resource occupied
-3 [MBC_PERM]	Permission denied

© 2025 Robox SpA

Syntax	**i32 mbcu_write_coils (i32 mbc, i32 unitID, i32 offsCoilStart, i32 nCoil, source)**

Value	Meaning
-4 [MBC_CONN]	Connection failed
-5 [MBC_NOBUFS]	Buffer exhausted
-6 [MBC_NOMEM]	Insufficient memory
-7 [MBC_TX_ERR]	Transmission error
-8 [MBC_RX_ERR]	Receive error
-9 [MBC_TIMEOUT]	Blocking command timeout
-10 [MBC_EXC_CODE_1]	Response with exception 1 (function code not supported by server)
-11 [MBC_EXC_CODE_2]	Response with exception 2 (invalid address)
-12 [MBC_EXC_CODE_3]	Response with exception 3 (invalid quantity or value)
-13 [MBC_EXC_CODE_4]	Response with exception 4 (operation failed)
-14 [MBC_WRONG_ANSWER]	Invalid response
-15 [MBC_INVAL_MBC]	Invalid *mbc* id

Validity	Task
Note	The function is blocking. Exits only when the task is completed

Function mbcu_write_multiple_coils()

This function writes COILS to a Modbus server by specifying the unit ID and using the Modbus function code of the corresponding multiple writes FC 15, unlike mbcu_write_coils 405() which can also use FC 05.

Syntax	**i32 mbcu_write_multiple_coils (i32 mbc, i32 unitID, i32 offsCoilStart, i32 nCoil, source)**
mbc	Handle value returned by mbc_init 366()

Syntax	**i32 mbcu_write_multiple_coils (i32 mbc, i32 unitID, i32 offsCoilStart, i32 nCoil, source)**
unitID	Value of the unit ID to be used
offsCoilStart	Initial index of the COIL
nCoil	Number of COILS to be written. **NOTE**: The maximum number of COILS that can be written is 1968.
source	Area where to take the data to be written. Can be i16/u16/i32 or array of i16/u16/i32
Result	Returns:

Value	Meaning
0 [MBC_OK]	Command executed
-1 [MBC_INVAL]	Invalid arguments
-2 [MBC_BUSY]	Resource occupied
-3 [MBC_PERM]	Permission denied
-4 [MBC_CONN]	Connection failed
-5 [MBC_NOBUFS]	Buffer exhausted
-6 [MBC_NOMEM]	Insufficient memory
-7 [MBC_TX_ERR]	Transmission error
-8 [MBC_RX_ERR]	Receive error
-9 [MBC_TIMEOUT]	Blocking command timeout
-10 [MBC_EXC_CODE_1]	Response with exception 1 (function code not supported by server)
-11 [MBC_EXC_CODE_2]	Response with exception 2 (invalid address)
-12 [MBC_EXC_CODE_3]	Response with exception 3 (invalid quantity or value)
-13 [MBC_EXC_CODE_4]	Response with exception 4 (operation failed)
-14 [MBC_WRONG_ANSWER]	Invalid response

© 2025 Robox SpA

Syntax	**i32 mbcu_write_multiple_coils (i32 mbc, i32 unitID, i32 offsCoilStart, i32 nCoil, source)**
	Value: -15 [MBC_INVAL_MBC] — Meaning: Invalid *mbc* id
Validity	Task
Note	The function is blocking. Exits only when the task is completed

Function mbcu_write_multiple_regs()

This function writes HOLDING REGISTERS to a modbus server by specifying the unit ID and using the Modbus function code of the corresponding multiple writes FC 16, unlike mbcu_write_reg 410() which can also use FC 06.

Syntax	**i32 mbcu_write_multiple_regs (i32 mbc, i32 unitID, i32 offsHRStart, i32 nHR, i32 endian, source)**
mbc	Handle value returned by mbc_init 366()
unitID	Value of the unit ID to be used
offsHRStart	Initial index of the HOLDING REGISTER
nHR	Number of HOLDING REGISTERs to be written. **NOTE**: The maximum number of HOLDING REGISTERs that can be written is 123.
endian	Type of endian to be used:

Value	Meaning
0 [EM_B_16]	Big Endian in the single word. (default)
1 [EM_L_16]	Little Endian in the single word
2 [EM_B_32]	Big Endian in 32 bits
3 [EM_L_32]	Little Endian in 32 bits
4 [EM_MB_32]	Middle Big Endian in 32 bits
5 [EM_ML_32]	Middle little Endian in 32 bits
6 [EM_B_64]	Big Endian in 64 bits
7 [EM_L_64]	Little Endian in 64 bits

© 2025 Robox SpA

Syntax	**i32 mbcu_write_multiple_regs (i32 mbc, i32 unitID, i32 offsHRStart, i32 nHR, i32 endian, source)**

Value	Meaning
8 [EM_MB_64]	Middle Big Endian in 64 bits
9 [EM_ML_64]	Middle little Endian in 64 bits

source	Area where to take data to write. Can be i16/u16/i32/float/real or array of i16/u16/i32/float/real
Result	Returns:

Value	Meaning
0 [MBC_OK]	Command executed
-1 [MBC_INVAL]	Invalid arguments
-2 [MBC_BUSY]	Resource occupied
-3 [MBC_PERM]	Permission denied
-4 [MBC_CONN]	Connection failed
-5 [MBC_NOBUFS]	Buffer exhausted
-6 [MBC_NOMEM]	Insufficient memory
-7 [MBC_TX_ERR]	Transmission error
-8 [MBC_RX_ERR]	Receive error
-9 [MBC_TIMEOUT]	Blocking command timeout
-10 [MBC_EXC_CODE_1]	Response with exception 1 (function code not supported by server)
-11 [MBC_EXC_CODE_2]	Response with exception 2 (invalid address)
-12 [MBC_EXC_CODE_3]	Response with exception 3 (invalid quantity or value)
-13 [MBC_EXC_CODE_4]	Response with exception 4 (operation failed)

© 2025 Robox SpA

Syntax	**i32 mbcu_write_multiple_regs (i32 mbc, i32 unitID, i32 offsHRStart, i32 nHR, i32 endian, source)**
	Value / Meaning: -14 [MBC_WRONG_ANSWER] — Invalid response -15 [MBC_INVAL_MBC] — Invalid *mbc* id
Validity	Task
Note	The function is blocking. Exits only when the task is completed

Function mbcu_write_reg()

This function writes HOLDING REGISTERS to a modbus server specifying the unit ID.

Syntax	**i32 mbcu_write_reg (i32 mbc, i32 unitID, i32 offsHRStart, i32 nHR, i32 endian, source)**
mbc	Handle value returned by mbc_init 366()
unitID	Value of the unit ID to be used
offsHRStart	Initial index of the HOLDING REGISTER
nHR	Number of HOLDING REGISTERs to be written. **NOTE**: The maximum number of HOLDING REGISTERs that can be written is 123.
endian	Type of endian to be used: Value / Meaning: 0 [EM_B_16] — Big Endian in the single word. (default) 1 [EM_L_16] — Little Endian in the single word 2 [EM_B_32] — Big Endian in 32 bits 3 [EM_L_32] — Little Endian in 32 bits 4 [EM_MB_32] — Middle Big Endian in 32 bits 5 [EM_ML_32] — Middle little Endian in 32 bits 6 [EM_B_64] — Big Endian in 64 bits

© 2025 Robox SpA

Syntax	**i32 mbcu_write_reg (i32 mbc, i32 unitID, i32 offsHRStart, i32 nHR, i32 endian, source)**

Value	Meaning
7 [EM_L_64]	Little Endian in 64 bits
8 [EM_MB_64]	Middle Big Endian in 64 bits
9 [EM_ML_64]	Middle little Endian in 64 bits

source	Area where to take data to write. Can be i16/u16/i32/float/real or array of i16/u16/i32/float/real
Result	Returns:

Value	Meaning
0 [MBC_OK]	Command executed
-1 [MBC_INVAL]	Invalid arguments
-2 [MBC_BUSY]	Resource occupied
-3 [MBC_PERM]	Permission denied
-4 [MBC_CONN]	Connection failed
-5 [MBC_NOBUFS]	Buffer exhausted
-6 [MBC_NOMEM]	Insufficient memory
-7 [MBC_TX_ERR]	Transmission error
-8 [MBC_RX_ERR]	Receive error
-9 [MBC_TIMEOUT]	Blocking command timeout
-10 [MBC_EXC_CODE_1]	Response with exception 1 (function code not supported by server)
-11 [MBC_EXC_CODE_2]	Response with exception 2 (invalid address)
-12 [MBC_EXC_CODE_3]	Response with exception 3 (invalid quantity or value)
-13 [MBC_EXC_CODE_4]	Response with exception 4 (operation failed)

© 2025 Robox SpA

Syntax	**i32 mbcu_write_reg (i32 mbc, i32 unitID, i32 offsHRStart, i32 nHR, i32 endian, source)**
	Value / Meaning -14 [MBC_WRONG_ANSWER] — Invalid response -15 [MBC_INVAL_MBC] — Invalid *mbc* id
Validity	Task
Note	The function is blocking. Exits only when the task is completed

Function minimum()

Function that returns the minor of the two values given as input.

Syntax	**real minimum (real val1, real val2)**
val1	First value to compare
val2	Second value to compare
Result	Returns the lesser of *val1* and *val2*
Validity	Rule, Task
Note	

Examples

Example of use:

```
velCalc = 60
velMin = 30

vel = minimum(velCalc, velMin)
; vel = 30
```

Function mod()

Function that returns the remainder of the division between numeric variables.

Syntax	**real mod (real dividend, real divisor)**
dividend	Value whose module is desired
divisor	Module value

© 2025 Robox SpA

Syntax	**real mod (real dividend, real divisor)**
Result	Returns the remainder of the division, calculated using the following logic: mod = *dividend* - *(divisor* * (int)*(dividend / divisor*))
Validity	Rule, Task
Note	

Examples

Example of use:

```
r(11) = 20
r(12) = mod(r(11), 16)
; After execution r(12) will contain 4, i.e., 4 = (20 - 16 * 1)

qMod = mod(ip(master), stepAx2)
; ip(master) ranges from 0 to infinity while qMod ranges from 0 to
stepAx2
```

Function mot_append()

Function to add data to the MOT points set.

Syntax	**i32 mot_append (i32 motId, real posX, real posY, real velY, real accY [, real other=NULL])**
motId	MOT reference code (obtained from the mot_create() function)
posX	Value of the interpolation key associated with the point (abscissa, x-coordinate). Depending on the flag value used in mot_create(), the value of *posX* must be mandatorily higher than the last one entered or it can take any value
posY	Array containing the data of the n ordinates (position coordinates relative to the *posX* abscissa)
velY	Array containing the data of the n first derivatives (velocity coordinates relative to the *posX* abscissa). **WARNING**: These are required if the interpolation type is equal to 3 or 5. Otherwise, specify NULL.
accY	Array containing the data of the n second derivatives (acceleration coordinates relative

© 2025 Robox SpA

Syntax	**i32 mot_append (i32 motId, real posX, real posY, real velY, real accY [, real other=NULL])**
	to the *posX* abscissa). **WARNING**: These are required if the interpolation type is equal to 5. Otherwise, specify NULL.
other	Data structure containing auxiliary data. (optional, default NULL) **NOTE**: A structure must necessarily be used to handle auxiliary data. If mixed type was chosen as interpolation (flag = 0xF0000, see mot_create()), the first field of the *other* structure must be an i32 specifying the type of interpolation of the segment (1=linear, 2=automatic cubic, 4=automatic pent). There are no restrictions on the other fields in the structure.
Result	Returns:

Value	Meaning
1	Unable to enter point, container full
0	Operation OK
-1	*motID* does not exist
-2	Key less than or equal to the last one introduced and insertion flag absent
-3	Illegal *posY* value (data size smaller than expected)
-4	Illegal *other* value (size different from that specified at creation)
-5	Internal error
-6	MOT manager busy (other transaction in progress)
-10	Missing data from the first derivative
-11	Data missing from derivative second

© 2025 Robox SpA

Syntax	**i32 mot_append (i32 motId, real posX, real posY, real velY, real accY [, real other=NULL])**
	<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>-12</td><td>Type of interpolation not handled</td></tr></table>
Validity	Rule, Task
Note	

Examples

Example of use:

© 2025 Robox SpA

```
function int writeMotParameter(int idMot, int numPointsCam, int
numParameters, real velRef)

; -------- WARNING ---------
; use this function only if you have extracted the cam points from a
file organised with the
; master position in the first coloumn, the slave position int the
second and, eventually,
; the slave speed in the third and the slave acceleration in the
fourth
; --------------------------

; this function insert the cam points previusly extracted from a file
and stored in the data[] array

; numPointsCam is equal to the number of row of the file
; numParameters is equal to the number of coloumn of the file

int idx
real velRef, velXrefQ, posX, posY, velY, accY

idx = 0
velXrefQ = velRef * velRef
; erase previous data
reso = mot_clear(idMot)
if(reso < 0)
    return -1
endif

str = "Adding points to the MOT " # str_i(idMot, 10, 16)
inf_report(0x0100, str)

; Data saved
;-------------------------------------------

; in this example the master is assumed to go from 0 to 360 degree
; speed and acceleration data usually must be normalized, this is why
they
; are divided to velRef and velXrefQ

; initial data before 0 deg equal to previous cycle end, to well
calculate derivatives
for(k = 4, k > 0, k--)

    idx =(numPointsCam - (k+1)) * numParameters

    posX = data[idx] - rMasterCycle
    posY = data[idx+1]
    velY = data[idx+2] / velRef    ; [mm/deg]
    accY = data[idx+3] / velXrefQ ; [mm/deg^2]

    select(numParameters)

        case 2
            reso = mot_append(idMotot, posX, posY, NULL, NULL)
            break

        case 3
            reso = mot_append(idMot, posX, posY, velY, NULL)
            break

        case 4
```

Function mot_clear()

Function to delete all data in the MOT point set.

Syntax	**i32 mot_clear (i32 motId)**
motId	MOT reference code (obtained from the mot_create 417() function)
Result	Returns: Value / Meaning 0 / Operation ok -1 / *motID* does not exist -5 / Internal error -6 / MOT manager busy (other operation in progress)
Validity	Rule, Task
Note	

Function mot_create()

Function to create the MOT. The memory area for managing the set of points is created.

Syntax	**i32 mot_create (i32 nEl, i32 nInt, i32 sizeOfOther, i32 flag)**
nEl	Size in number of points of the MOT (>=1)
nInt	Number of units to be interpolated (ordered) for each point (>= 1)
sizeOfOther	Size in bytes of auxiliary information associated with the point (>=0)
flag	Management flag: Bit / Hexadecimal value / Meaning 0 / 0x00000001 / If TRUE, allows insertion of points with any abscissa instead of increasing abscissa (see mot_append 413())

Syntax	**i32 mot_create (i32 nEl, i32 nInt, i32 sizeOfOther, i32 flag)**

Bit	Hexadecimal value	Meaning
1	0x00000002	If TRUE, the additional data returned by mv_mot_exec() is referenced to the end point instead of the starting point of the interpolation segment
4	0x00000010	If TRUE, deletes points after execution (point (n-1) is deleted when execution passes point n)
Interpolation type value code (Bit 16-:-19, 0x000F00xx):		
16	0x000100xx	Linear interpolation
17	0x000200xx	Cubic interpolation with automatic calculation of parameters
16-17	0x000300xx	Cubic interpolation calculated by specifying the value of the first derivative at the point (the velocity, if interpolating positions)
18	0x000400xx	Pentic (fifth-order) interpolation with automatic calculation of parameters
16-18	0x000500xx	Pentic interpolation calculated by specifying the values of the first and second

Syntax	**i32 mot_create (i32 nEl, i32 nInt, i32 sizeOfOther, i32 flag)**

Bit	Hexadecimal value	Meaning
		derivatives at the point (the velocity and acceleration, if it is interpolation of positions)
16-17-18-19	0x000F00xx	Mixed interpolation (different for each segment). The type is specified in the *other* structure (see mot_append 413 ()). The first field of the *other* structure used in mot_append() must necessarily be of type i32 and contain the type of interpolation: 1=linear, 2=automatic cubic, 4=automatic pent

Result

Returns:

Value	Meaning
> 0	OK operation. The value represents the code of the MOT
-1	Illegal *nEl* value
-2	Value *nInt* illegal
-3	Excessive *nEl* value, insufficient memory
-4	Internal error
-8	Selected mixed interpolation, but *other* structure does not

© 2025 Robox SpA

Syntax	**i32 mot_create (i32 nEl, i32 nInt, i32 sizeOfOther, i32 flag)**
	Value / Meaning
	exist or has *size* < *sizeOfOther*
	-9 — Unknown interpolation code
Validity	Rule, Task
Note	

NOTE: Each memory zone created with mot_create() must be destroyed with a corresponding mot_delete 421(); next calls to mot_create() create different memory zones.

Examples

Example of use:

```
int flag, flag2
string appo

; MOT 1
;-----------------------------------------------------------------
flag.(0) = 1    ; data input enable
flag = flag r_or (0x50000)    ; pentic with speed and acc data
rMotID = mot_create 417(1000, 1, 0, flag)
if(rMotID < 0)
    str_format 601(appo, "Unable to create MOT error code :%d", rMotID)
    inf_report 697(0x100, appo)
    rMotID = 0
endif

; MOT 2
;-----------------------------------------------------------------
flag2.(0) = 1    ; data input enable
flag2 = flag2 r_or (0x40000)    ; pentic without speed and acc data
rMotID2 = mot_create(1000, 1, 0, flag2)
if(rMotID2 < 0)
    str_format(appo, "Unable to create MOT2 error code :%d", rMotID2)
    inf_report(0x100, appo)
    rMotID2 = 0
endif
```

© 2025 Robox SpA

Function mot_delete()

Function to delete the MOT previously generated with mot_create 417(). The MOT memory area labeled *motId* is cleared.

Syntax	**i32 mot_delete (i32 motId)**
motId	MOT reference code (obtained from the mot_create 417() function)
Result	Returns: Value / Meaning 0 / Operation ok -1 / *motID* does not exist -5 / Internal error -6 / MOT manager busy (other operation in progress)
Validity	Rule, Task
Note	

Function mot_get()

Function to read data related to a point on an MOT.

Syntax	**i32 mot_get (i32 motId, i32 nPoint, real posX, real posY=NULL, real velY=NULL, real accY=NULL [, other=NULL])**
motId	MOT reference code (obtained from the mot_create 417() function)
nPoint	Point number (0 = first point, -1 = last point)
posX	Value (returned by the function) of the interpolation key associated with the point (abscissa, x-coordinate)
posY	Array of doubles intended to hold the data (returned by the function) of the n ordinates (position coordinates relative to the *posX* abscissa). Specify NULL (default) if you do not want to get the data. If specified <>NULL, the mot_get() function will download the *nPoint* data into the array
velY	Array of doubles intended to hold the data (returned by the function) of the n velocities (velocity coordinates relative to the *posX* abscissa).

© 2025 Robox SpA

Syntax	**i32 mot_get (i32 motId, i32 nPoint, real posX, real posY=NULL, real velY=NULL, real accY=NULL [, other=NULL])**
	Specify NULL (default) if you do not want to get the data. If specified <>NULL, the mot_get() function will download the *nPoint* point data into the array
accY	Array of doubles intended to hold the data (returned by the function) of the n accelerations (acceleration coordinates relative to the *posX* abscissa *)*. Specify NULL (default) if you do not want to get the data. If specified <>NULL, the mot_get() function will download the *nPoint* data into the array
other	Data structure intended to hold the auxiliary data (returned by the function). (optional, default NULL) If specified, the mot_get() function downloads into the array the data of the *nPoint*
Result	Returns:

Value	Meaning
1	Unable to enter point, container full
0	Operation OK
-1	*motID* does not exist
-2	Illegal *nPoint* value
-3	Illegal *posY* value (smaller data size than expected)
-4	Illegal *other* value (size different from the size specified at creation)
-6	MOT manager busy (other operation in progress)
-7	Empty (no point)

Validity	Rule, Task
Note	

© 2025 Robox SpA

Function mot_remove()

Function to delete one or more points. Deletes all points having distance <= *epsi* from the specified *posX*.

Syntax	**i32 mot_remove (i32 motId, real posX [, real epsi=1e-7])**
motId	MOT reference code (obtained from the mot_create 417() function)
posX	Value of the interpolation key associated with the point to be removed (abscissa, x-coordinate)
epsi	Range of the deletion. (optional, default 1e-7)
Result	Returns:

Value	Meaning
>= 0	Number of points eliminated
-1	*motID* does not exist
-5	Internal error
-6	MOT manager busy (other operation in progress)

Validity	Rule, Task
Note	

Function mt_ats()

Useful function for constructing an atomic semaphore.

Note: Deprecated. The use of mt_ats2 424() is recommended.

Syntax	**i32 mt_ats (i32 var [, i32 value=1])**
var	Variable used as a semaphore
value	Value to be imposed (optional, default 1). Use *value<>0* to get the semaphore, *value=0* to release the semaphore
Result	Returns:

Value	Meaning
0	Semaphore already busy

© 2025 Robox SpA

Syntax	**i32 mt_ats (i32 var [, i32 value=1])**
	Value: 1 — Meaning: Semaphore clear and properly engaged by the function
Validity	Rule, Task
Note	

Examples

Example of use:

```
int mySem
if(mt_ats(mySem))
    ; Free semaphore: I write the data atomically
    ...
    mySem = 0   ; release the semaphore
else
    ; the semaphore is busy: let's wait
end_if
```

Function mt_ats2()

Useful function for constructing an atomic semaphore.

Syntax	**i32 mt_ats2 (i32 var [, i32 value=1])**
var	Variable used as a semaphore
value	Value to be imposed (optional, default 1). Use *value<>0* to get the semaphore, *value=0* to release the semaphore
Result	Returns: Value: <> 0 — Meaning: Semaphore already busy. Represents the value of the variable from the previous entry Value: 0 — Meaning: Semaphore free and properly used by the function
Validity	Rule, Task
Note	

© 2025 Robox SpA

Examples

Example of use:

```
int mySem
int result

if(mt_ats2(mySem)=0)
    rr(23) = rr(49)
    rr(24) = rr(52)
    ...
    ; releasing the semaphore
    result = mt_ats2(mySem, 0)
else
    ; someone else is using the semaphore, let's do something else
    ...
endif
```

Function mv_cam()

Function to perform a third-order polynomial.

Two cams can be set, and through an appropriate *par* parameter it is possible to switch between cams.

Functioning

Depending on an input unit (master), the corresponding value of the slave unit is calculated using a table of information (the result, for example, of spreadsheets or mathematical programs) that has been previously loaded onto the control flash-card(f_read_table_s()).

By optional parameters, it is possible to perform the transition between two polynomials or the weighted sum of them.

Syntax	**real mv_cam (real xCurr, i32 nSegm1, stru_matcam stru1 [, real par=0] [, i32 nSegm2=0] [, stru_matcam stru2=NULL])**
xCurr	Current master value (abscissa)
nPoints1	Number of points of which the first polynomial cam is composed. Must match the length of the array of *stru1*
stru1	Array of structures of type stru_matcam containing the values of the first cam. **NOTE**: Must be initialized as an array.
par	Mixer parameter that allows switching between the two cams. (optional, default 0)

Syntax	**real mv_cam (real xCurr, i32 nSegm1, stru_matcam stru1 [, real par=0] [, i32 nSegm2=0] [, stru_matcam stru2=NULL])**
	<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>Performs cam 1</td></tr><tr><td>0 -:- 1 (extremes not included)</td><td>Performs a weighted overlap of the two cams</td></tr><tr><td>1</td><td>Performs cam 2</td></tr></table>
nPoints2	Number of points of which the second polynomial cam is composed. (optional, default 0) Must match the length of the array of *stru2*
stru2	Array of structures of type stru_matcam 846 containing the values of the second cam. (optional, default NULL) **NOTE**: Must be initialized as an array.
Result	Updated result value, corresponding to the y-coordinates of the cam. The result is calculated as: dx1 = *xCurr* - xInitial1 dx2 = *xCurr* - xInitial2 result = (((a1*dx1^3) + (b1*dx1^2) + (c1*dx1) + d1) * (*1-par*)) + (((a2*dx2^3) + (b2*dx2^2) + (c2*dx2) + d2) * *par*)
Validity	Rule
Note	Parameters a, b, c, d must be given as input to the function

Diagnostics

- Generates alarm 9900 ill. Arg A:5 T:x ST:xxx if *par* <> 0 and *nSegm2* = 0

Examples

Example of use:

```
IP(AX_2) = mv_cam 425(IP(1), 5, traiettoria)
```

For a concrete example see in the RDE environment: Workspace -> Specials -> Examples Library.

Function mv_crimper()

A function that is used to realize synchronized motion, in which one axis (slave) moves continuously related to the quota of another axis (master).

Functioning

The movement of the slave includes a first phase (tracking) in which its speed is proportional to that of the master, followed by a second phase (recovery) during which the slave performs a movement such that, at the end of the cycle, it reappears in the right phase position and with the right speed in order to start, without perturbation, the next cycle.

The recovery motion is calculated by using a sinusoidal law of motion, with peak velocity calculated as a function of the working parameters. The master axis can be a controlled axis, or an external reference (e.g., a line encoder) or even a virtual quota constructed and evolved by the R3 application software.

The slave axis must necessarily be a controlled axis.

Operationally, the function must be invoked, with the master stopped, with the code M_INITIALIZING and initialize all the function parameters, then perform a positioning on the quota returned by the function and then enable the movement of the master.

Syntax	**real mv_crimper (i32 return, stru_crimper stru)**
return	Variable containing information about the state of the function (see returns codes 61).

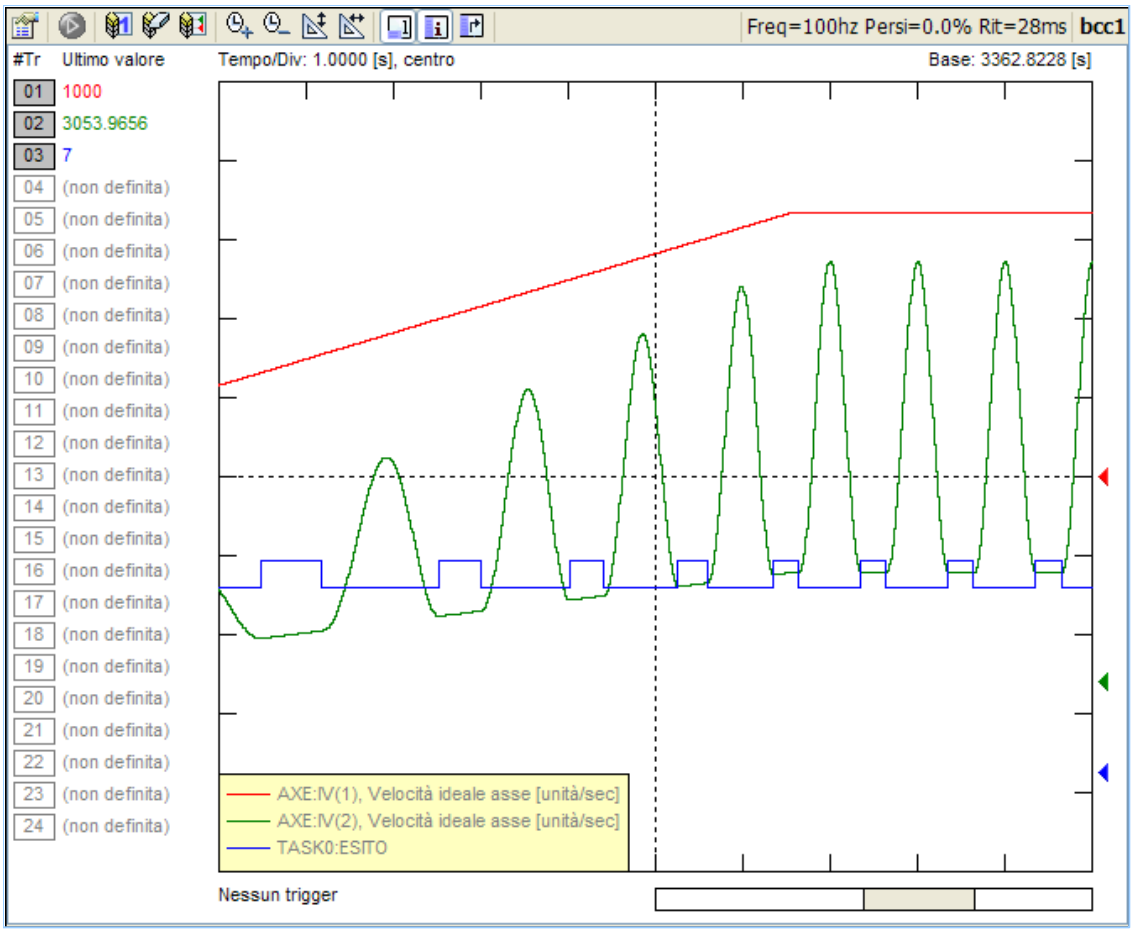
Value	Description
1 [M_INITIALING]	When the function is first called, the user must set the state to M_INITIALING
7 [M_HOMING]	The recovery mission is in progress
8 [M_SYNCHRONIZED]	The slave is tracking the master

stru	Structure of type stru_crimper 812 containing parameters related to the movement
Result	Updated position value of the slave axis
Validity	Rule
Note	

Examples

Example on oscilloscope of a possible output of the function:

© 2025 Robox SpA



For a concrete example see in the RDE environment: Workspace -> Specials -> Examples Library.

Function mv_follow()

Function that is used to realize tracking of one axis (slave) against another (master).

HINT: We recommend using the more performant mv_follow2 432() function.

Functioning

The slave axis is essentially handled by 4 successive steps (see stru_follow 816 for keyword description):

- M_POSITIONING: the slave axis moves to the *qParkSlave* position, using the velocity parameters *velMaxSlave* and acceleration *accMaxSlave*.
- M_WAITING: the axis stands still. The right moment to start tracking is computed.Conditions allowing successful tracking are evaluated (master direction and speed, *qSyncMaster* position compared to the master current position, *qsyncSlave* position compared to the current slave position). If the slave acceleration is not sufficient the code M_LOW_ACCELERATION is issued.
- M_FOLLOWING: the axis moves in the direction specified by the *slaveDirection* parameter, performing a constant acceleration motion calculated to achieve the master

axis speed at the slave quota *qSyncSlave* and the master quota *qSyncMaster*. Tracking also considers any changes in master speed during the cycle.

- M_REACHED: the slave axis keeps tracking with the master, based on its current speed, until the slave quota *qSyncOut* has been reached or exceeded. At this point, the unit *qSyncMaster is* incremented by *masterCycle* and units *qPark*, *qSyncSlave*, *qSyncOut* are incremented by *slaveCycle,* to prepare for the next cycle. The phase variable is set to M_POSITIONING and the cycle starts again.

The *kSlaveMaster* parameter results very useful in those cases where the unit of measurement of the slave axis is different from that of the master axis (typical case a master expressed in mm and a slave in degrees). By specifying the K factor existing between the two axes, it is possible to perform tracking as if the units were homogeneous with each other.

If cyclic tracking with monodirectional axes is desired (e.g., master conveyor, slave two-blade cutting jaw), a step value for both axes must be specified (in the example, *MasterCycle* = product length, *SlaveCycle* = 180 degrees). These values are used by the function to automatically increment the synchronism (master and slave), end tracking quota, and wait quota. The increments all occur at the same time when the synchronism phase is exited.

If, on the other hand, the slave axis has reciprocating motion (e.g., no-end screw cutting axis), *SlaveCycle* = 0 must be specified for it. In this way, at the end of the cycle, the slave axis returns to the *qPark* position (which is the same for all cycles) while the next synchronism on the master axis*(qSyncMaster*) is incremented.

If "one shot" tracking is desired, that is, without automatic repetition, simply specify *MasterCycle* = 0. This way, once the first tracking cycle is finished, the conditions for starting with a new cycle do not reoccur until the user directly enters an updated *qSyncMaster* value.

The first time the function is called, it is necessary to have previously loaded a set of appropriate values into the stru_follow structure. The *status* parameter must also have been initialized to M_POSITIONING (initial positioning). This parameter can be usefully consulted by the R3 application software during the course of the cycle. It is the responsibility of the application software to ensure that when the function is called, the tables contain consistent data.

Syntax	**real mv_follow (i32 return, stru_follow stru)**
return	Variable containing information about the state of the function (see returns codes 61).

Value	Description
2 [M_POSITIONING]	When the function is first called, the user must set the state to M_POSITIONING. The slave axis is moving to the parking position
3 [M_WAITING]	The slave axis is waiting to start
4 [M_WAITING_NEXT_CYCLE]	There are no conditions to start.

© 2025 Robox SpA

Syntax	**real mv_follow (i32 return, stru_follow stru)**
	Value: Description : Waiting for the next appointment 6 [M_REACHED]: The slave axis is tracking
stru	Structure of type stru_follow 816 containing parameters related to movement
Result	Updated value of slave axis position
Validity	Rule
Note	

Following can be single or cyclically repeated, but in any case there must be a waiting phase beginning tracking, in which the axis is stationary.

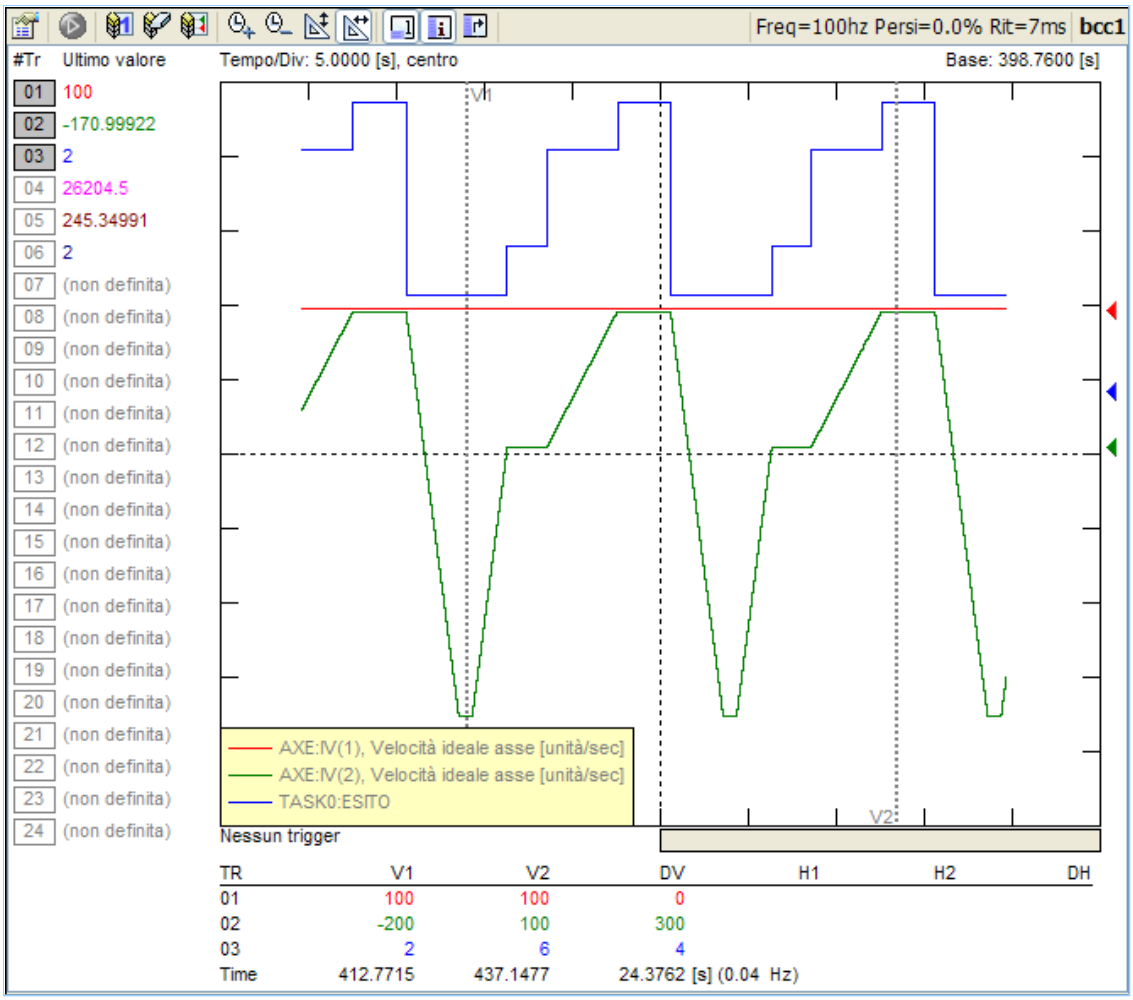
The master could be a controlled physical axis, but also a simple external reference (e.g., a line encoder) or even a virtual quota constructed and evolved by the application software.

The slave axis must necessarily be a controlled axis.

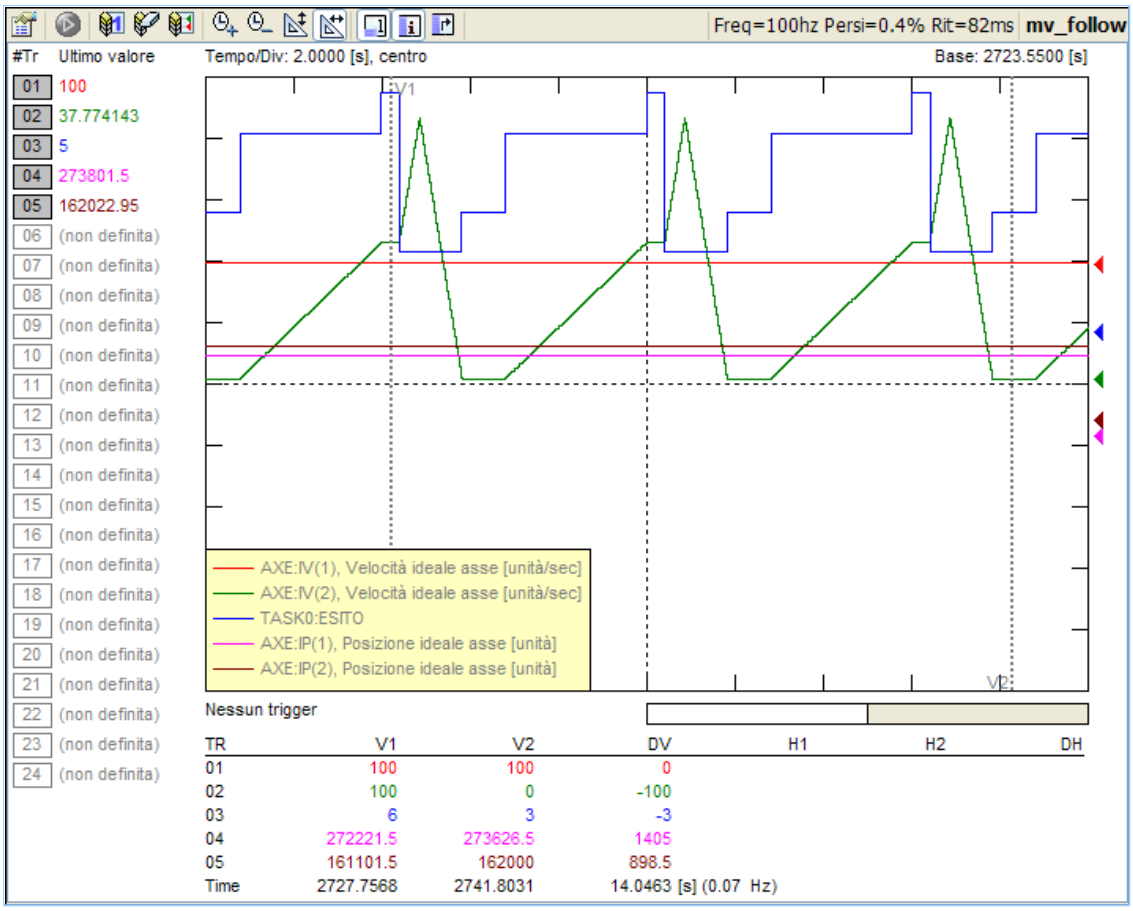
Examples

Example on oscilloscope of a linear tracking (slave goes back and forward):

© 2025 Robox SpA



Example on oscilloscope of a cyclic tracking (slave with rotational motion):



For a concrete example see in the RDE environment: Workspace -> Specials -> Examples Library.

Function mv_follow2()

This function is used to perform tracking of a SLAVE against a MASTER, both not associated with an axis. The function performs jerked movements computed with a pentic curve (5th order), cam of the position of the master axis.

Premise

Tracking can be single or cyclically repeated in fixed cycle or variable cycle.

The master could be a controlled physical axis, but also a simple external reference (e.g., a line encoder) or even a virtual quota constructed and evolved by the application software.

In any case, during the execution of the function, the master must always evolve in the same direction.

The slave could be a controlled physical axis, but also a simple unit built and evolved by the application software. It could be a linear axis of limited excursion or a cyclic axis (e.g., rotary).

Types of the follow2 function

The syntax of follow2 varies depending on:

- MASTER and SLAVE both not associated with an axis -> mv_follow2()

- MASTER not associated with an axis, SLAVE associated with an axis -> mva_follow2 442()
- MASTER and SLAVE both associated with an axis (typical case) -> mva_follow2_m 443()

Functioning

The slave (follower) axis is managed according to the following steps:

- M_POSITIONING: the initial positioning step must be requested by the user by setting the appropriate bits of the *command* parameter of the reference structure. The positioning request must necessarily be invoked from a condition where the slave is stationary in both velocity and acceleration (check that the ax_in_mov2 mask bit of the corresponding axis is 0). If positioning to the parking quota has been requested, the slave axis moves to the *qParkSlave* position, using the kinematics parameters *velPositSlave*, *accPositSlave*, and *jerkPositSlave*. In the case that the movement of the function is interrupted (for example, due to an emergency condition), if the master has not moved that far in the meantime, positioning can be performed at the cycle position relative to the current quota of the master. Use the corresponding bit of the *command* parameter to do this. During positioning to the relative quota of the master, it is essential that the master is stationary. The user can also perform the positioning in a different way, computed by an external function, the important thing is that he takes care to bring the slave axis to the correct position before using the mv_follow2() function. If at the first execution of the function (with positioning request) the slave is already in the correct position, it is possible to leave the kinematics parameters *velPositSlave*, *accPositSlave* and *jerkPositSlave* at 0
- M_WAITING: axis remains stationary at the *qParkSlave* parking quota; calculations are performed to check if it is time to start tracking according to the current quota of the master, the *qSyncMaster* quota, and the positions set for the various slave phases. If it is too late to start during the current cycle, the M_WAITING phase switches to M_WAITING_NEXT_CYCLE
- M_FOLLOWING: the axis starts tracking by performing a calculated movement such that it reaches the speed of the master axis at the slave quota *qSyncSlave* and the master quota *qSyncMaster*. If the slave axis is linear, a simple jerk ramp is executed (in which the space traveled by the master is twice the space traveled by the slave). If, on the other hand, the slave axis is cyclic, the movement performed in this phase depends on all the other parameters that make the cycle; in this case the tracking ramp may also have over-elongations
- M_SYNCHRONIZED: the slave axis keeps tracking to the master, until the slave quota *qSyncOut* has been reached or exceeded. During this phase, the speed of the slave axis is given by the following formula: *vSlave* = *velMaster* * *kSlaveMaster*

At the end of the synchronized phase the behavior differs depending on how the stru_follow2 structure has been configured:

- In the case of a linear slave(*slaveCycle* = 0), it switches to the M_DECEL deceleration phase in which the slave slows down to a stop by traversing the *spaceStopSlave* braking space and executing a simple jerk ramp (space traversed by the master double the space traversed by the slave).
  At the end of the deceleration phase, the linear slave switches to the return phase to the parking position M_HOMING in which a bell-shaped velocity profile movement is executed such that the parking position *qParkSlave* is reached and maintained for the *spaceWaitInPark* duration before the next tracking phase begins
- In the case of a cyclic slave(*slaveCycle* > 0) with stopping at quota *qParkSlave* enabled during the cycle (see corresponding bit of the *command* parameter), it goes directly to the M_HOMING phase in which a calculated movement is performed so that the parking position *qParkSlave* is reached and held for the duration *spaceWaitInPark* before the next tracking phase begins

© 2025 Robox SpA

- In the case of a cyclic slave*(slaveCycle* > 0) with stopping at the *qParkSlave* elevation disabled during the cycle (see corresponding bit of the *command* parameter), it moves directly to the next tracking phase without stopping at the *qParkSlave* elevation. The movement is calculated to start at speed and reach the next synchronism always at speed at the *qSyncSlave* and *qSyncMaster* quotas. If the space missing for the master to reach the synchronism quota was significantly larger than the space missing for the slave, the slave could slow down and stop at an intermediate quota that could also be different from the quota *qParkSlave*

If the master cycle does not have a fixed duration (see the corresponding bit of the *command* parameter), it is necessary to set the new value of *qSyncMaster* in time at each cycle: the useful phases for setting the *qSyncMaster* synchronism quota relative to the next cycle are the tracking phase M_FOLLOWING and the synchronized phase M_SYNCHRONIZED relative to the current cycle.

If the quota *qSyncMaster* is set beyond the end of the synchronized phase M_SYNCHRONIZED the slave axis will go to the parking quota *qParkSlave*.

Also in the case of variable duration cycle masters, it is mandatory to set the *masterCycle* parameter such that a minimum duration of the master cycle is set. The minimum duration is used to limit the speed of the recovery phase and, in case the *qSyncMaster* quota is not set in time, to set the speed of the homing phase.

In case of a cyclic slave with dwell at the parking quota *qParkSlave* disabled during the cycle, it is possible to recover a *qSyncMaster* quota assigned beyond the time limit by enabling the corresponding bit of the *command* parameter. In this case, as soon as the new *qSyncMaster* synchronism quota is set, an on-the-fly connection will be performed (if possible) between the movement to the *qParkSlave* parking quota and the tracking movement to the new synchronism.

The *kSlaveMaster* parameter results very useful in those cases where the unit of measurement of the slave axis is different from that of the master axis (typical case a master expressed in mm and a slave in degrees). By specifying the K factor existing between the two axes, it is possible to perform tracking as if the units were homogeneous with each other. In addition, by slightly varying the *kSlaveMaster* parameter from its nominal value, it is possible to perform synchronism at different speeds (for example, if the slave axis is the jaw that welds and cuts the film of a wrapping machine, it is useful to reduce the speed of the slave axis relative to that of the master so as to avoid tearing the film during welding).

The first time the mv_follow2() function is called, it is necessary to have previously loaded a set of appropriate values into the stru_follow2 structure. In addition, in order to be certain of what the initial phase will be, a positioning request should be made via the appropriate bits of the *command* parameter.

If some parameters characterizing the loop are not consistent, the function returns the negative result M_IN_ALARM. In this case you can test the *errorCode* parameter to get an indication of what is wrong.

If the *maxSpe* and *maxAcc* parameters are set and if testing on the relative bits of the *command* parameter is not disabled, a check is made each time the mv_follow2() function is executed to avoid exceeding the set limits. If the limits are exceeded, the function returns the negative result M_IN_ALARM and executes a linear stop ramp with *maxAcc* acceleration. Again, the *errorCode* parameter can be consulted to see at what stage the speed and/or acceleration were exceeded.

Syntax	**i32 mv_follow2 (stru_follow2 stru1 [, real period=0 [, stru_follow2info stru2=NULL])**
stru1	Structure of type stru_follow2 819 containing the parameters related to the movement

© 2025 Robox SpA

Syntax	**i32 mv_follow2 (stru_follow2 stru1 [, real period=0 [, stru_follow2info stru2=NULL])**
period	Time interval between two consecutive executions of this function. (optional, default 0) If 0 is assumed to be equal to SI
stru2	Structure of type stru_follow2info 827 where to save ancillary (out) data. (optional, default NULL)
Result	Variable containing information about the status of the function (see returns codes 61)

Value	Description
2 [M_POSITIONING]	Initial positioning requested by setting the relative bits of the *command* parameter
3 [M_WAITING]	The slave axis is waiting to start
4 [M_WAITING_NEXT_CYCLE]	No conditions to start, waiting for next appointment
5 [M_FOLLOWING]	The slave axis has started tracking the master
8 [M_SYNCHRONIZED]	The slave axis has tracked and is synchronized to the master
17 [M_DECEL]	The slave axis has finished the synchronization phase and is decelerating
7 [M_HOMING]	The slave axis is returning to the parking point
-6 [M_IN_ALARM]	The function is in alarm, see *errorCode*

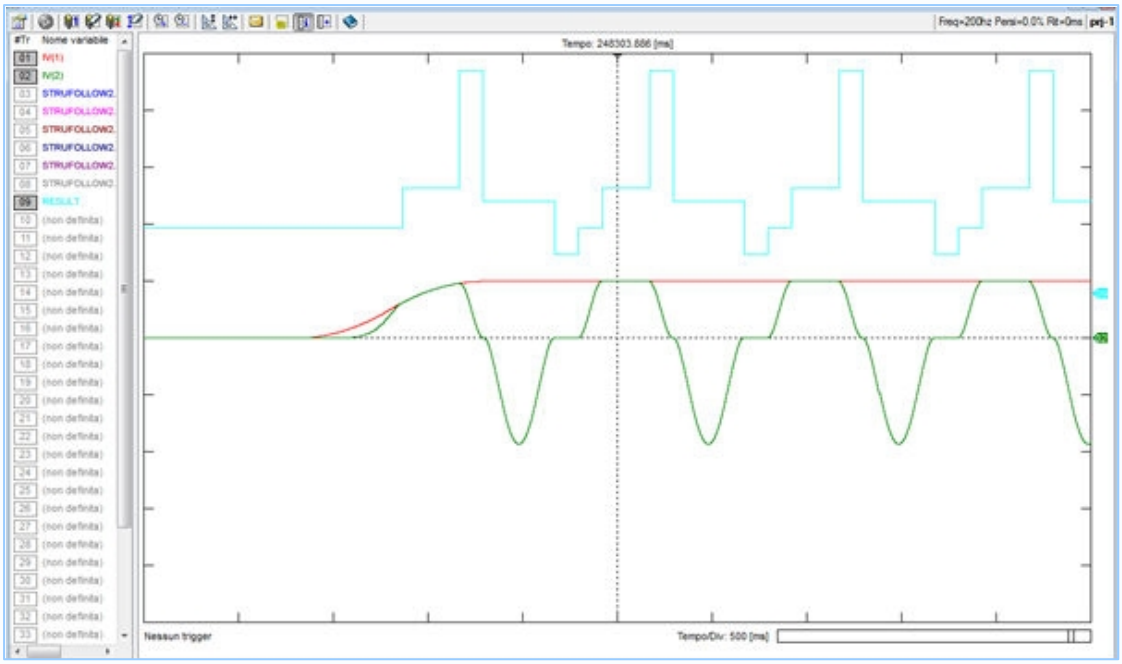
Validity	Rule
Notes	The configuration of the function is done through the auxiliary data structure

Examples

Example of use (fixed-loop master and linear slave):

© 2025 Robox SpA

```
stru_follow2 struFollow2

rule 1
motion
if(first_time321())
    struFollow2.command.0 = 0      ; 1 = force slave stop at
qParkSlave
 struFollow2.command.1 = 1  ; 1 = initial positioning to parking
quota
 struFollow2.command.2 = 0  ; 1 = initial positioning relative to
master loop
 struFollow2.command.16 = 0 ; 0 = fixed-cycle master, 1 = variable-
cycle master
 struFollow2.command.17 = 0 ; 1 = disable max_spe test
 struFollow2.command.18 = 0 ; 1 = disable max_acc test
 struFollow2.command.19 = 0 ; 1 = qSyncMaster aligned midway between
qSyncSlave and qOutSlave
 struFollow2.command.20 = 0 ; 1 = enable stop at qParkSlave (cyclic
slave)
 struFollow2.command.21 = 0 ; 1 = force slave movement forward only
(cyclic slave)
 struFollow2.command.22 = 0 ; 1 = enable on-the-fly sync recovery
 struFollow2.masterCycle = 400 ; [mm] Master fixed cycle of 400 mm
 struFollow2.qSyncMaster = 0 ; [mm] Master sync quota within the
cycle
 struFollow2.spaceWaitInPark = 50 ; [ mm] Master space with slave in
park.
 struFollow2.slaveCycle = 0 ; [mm] Slave cycle (0 = linear slave)
 struFollow2.qParkSlave = 0 ; [mm] Slave  parking quota
 struFollow2.qSyncSlave = 25 ; [mm] Quota of sync slave
 struFollow2.qQutSlave = 125 ; [mm] Quota of  end sync slave
 struFollow2.spaceStopSlave = 25 ; [mm] slave stop space
 struFollow2.velPositSlave = 100 ; [mm/s] positioning speed
 struFollow2.accPositSlave = 200 ; [mm/s^2] Positioning acceleration
 struFollow2.jerkPositSlave = 400 ; [mm/s^3] Jerk of positioning
 struFollow2.maxSpe = 1000 ; [mm/s] Maximum slave velocity
 struFollow2.maxAcc = 10000 ; [mm/s^2] Maximum slave acceleration
  struFollow2.kSlaveMaster = 1 ; k of master -> slave conversion
endif
struFollow2.qMaster    = ip(1)
struFollow2.velMaster = iv(1)
struFollow2.accMaster = ia(1)
struFollow2.qSlave     = ip(2)
struFollow2.velSlave   = iv(2)
struFollow2.accSlave   = ia(2)
result = mv_follow2432(struFollow2)
set_kinematics569(ax2, struFollow2.qSlave, struFollow2.velSlave,
struFollow2.accSlave)
end_motion
end_rule
```

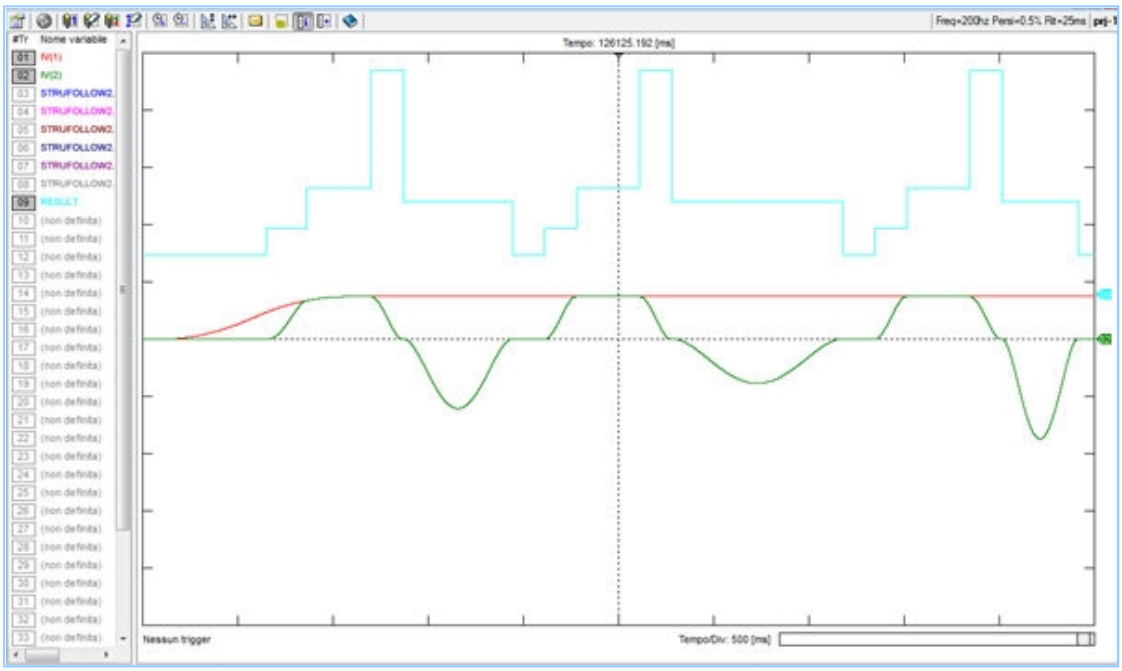


Example of use (variable cycle master and linear slave):

```
stru_follow2 struFollow2

rule 1
motion
if(first_time())
 struFollow2.command.0 = 0 ; 1 = force slave stop at qParkSlave
 struFollow2.command.1 =  1 ; 1 = initial positioning to parking
quota
 struFollow2.command.2 = 0 ; 1 = initial positioning relative to
master loop
 struFollow2.command.16 = 1 ; 0 = fixed-cycle master, 1 = variable-
cycle master
 struFollow2.command.17 = 0 ; 1 = disable max_spe test
 struFollow2.command.18 = 0 ; 1 = disable max_acc test
 struFollow2.command.19 = 0 ; 1 = qSyncMaster aligned midway between
qSyncSlave and qOutSlave
 struFollow2.command.20 = 0 ; 1 = enable stop at qParkSlave (cyclic
slave)
 struFollow2.command.21 = 0 ; 1 = force slave movement forward only
(cyclic slave)
 struFollow2.command.22 = 0 ; 1 = enable on-the-fly sync recovery
 struFollow2.masterCycle = 300 ; [mm] Variable cycle master (300 mm
is the minimum cycle)
 struFollow2.spaceWaitInPark = 50 ; [mm] Master space with slave in
park.
 struFollow2.slaveCycle = 0 ; [mm] Slave cycle (0 = linear slave)
```

© 2025 Robox SpA

```
 struFollow2.qParkSlave = 0 ; [mm] Slave  parking quota
 struFollow2.qSyncSlave = 25 ; [mm] Quota of sync slave
 struFollow2.qQutSlave = 125 ; [mm] Quota of  end sync slave
 struFollow2.spaceStopSlave = 25 ; [mm] slave stop space
 struFollow2.velPositSlave = 100 ; [mm/s] positioning speed
 struFollow2.accPositSlave = 200 ; [mm/s^2] Positioning acceleration
 struFollow2.jerkPositSlave = 400 ; [mm/s^3] Jerk of positioning
 struFollow2.maxSpe = 1000 ; [mm/s] Maximum slave velocity
 struFollow2.maxAcc = 10000 ; [mm/s^2] Maximum slave acceleration
 struFollow2.kSlaveMaster = 1 ; k of master -> slave conversion
endif
struFollow2.qMaster   = ip(1)
struFollow2.velMaster = iv(1)
struFollow2.accMaster = ia(1)
struFollow2.qSlave    = ip(2)
struFollow2.velSlave  = iv(2)
struFollow2.accSlave  = ia(2)
; Master sync quota derived from an OB of type rc_elementlocation
struFollow2.qSyncMaster = ip(1) + buffer.get_distance_to_end(0)
result = mv_follow2(struFollow2)
set_kinematics(ax2, struFollow2.qSlave, struFollow2.velSlave,
struFollow2.accSlave)
end_motion
end_rule
```

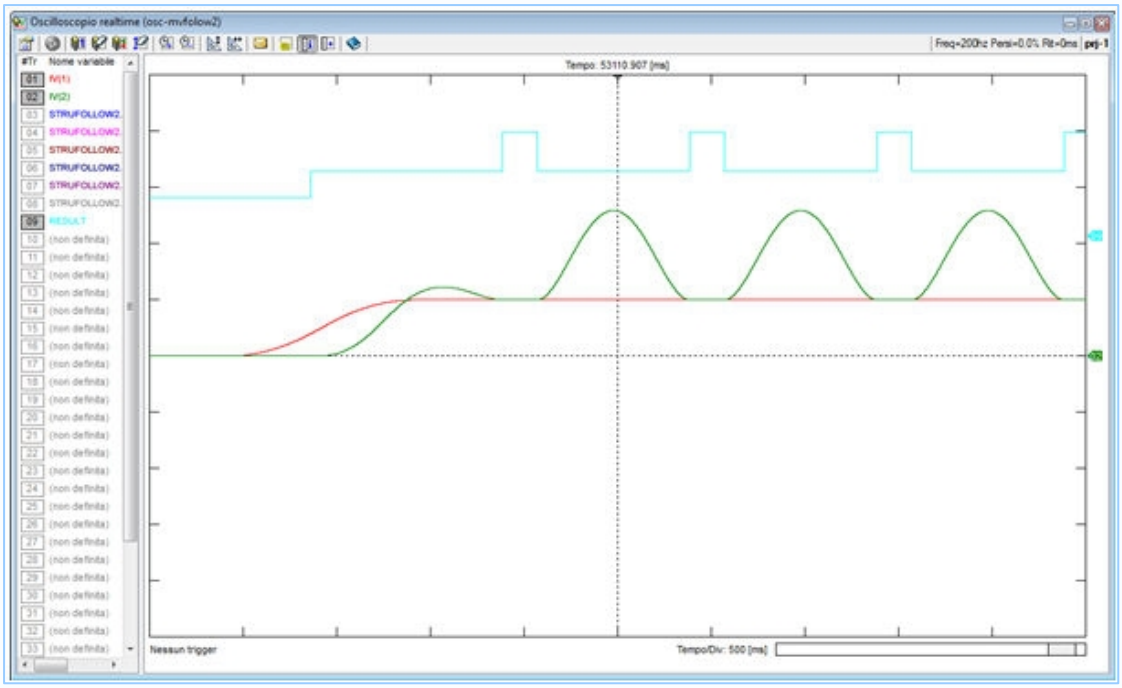


Example of use (fixed-cycle master and rotary slave):

```
stru_follow2 struFollow2

rule 1
```

```
motion
if(first_time())
   struFollow2.command.0 = 0 ; 1 = force slave stop at qParkSlave
 struFollow2.command.1 =  1 ; 1 = initial positioning to parking
quota
 struFollow2.command.2 = 0 ; 1 = initial positioning relative to
master loop
 struFollow2.command.16 =  0 ; 0 = fixed-cycle master, 1 = variable-
cycle master
 struFollow2.command.17 = 0 ; 1 = disable max_spe test
 struFollow2.command.18 = 0 ; 1 = disable max_acc test
 struFollow2.command.19 =  1 ; 1 = qSyncMaster aligned midway between
qSyncSlave and qOutSlave
 struFollow2.command.20 = 0 ; 1 = enable stop at qParkSlave (cyclic
slave)
 struFollow2.command.21 = 0 ; 1 = force slave movement forward only
(cyclic slave)
 struFollow2.command.22 = 0 ; 1 = enable on-the-fly sync recovery
 struFollow2.masterCycle = 400 ; [mm] Master fixed cycle of 400 mm
 struFollow2.qSyncMaster = 0 ; [mm] Master sync quota within the
cycle
 struFollow2.spaceWaitInPark = 0 ; [mm] Master space with slave in
park.
 struFollow2.slaveCycle = 360 ; [°] Slave cycle
 struFollow2.qParkSlave = 180 ; [°] Slave parking quota
 struFollow2.qSyncSlave = -20 ; [°] Quota of  slave sync
 struFollow2.qOutSlave = 20 ; [°] Quota of  end sync slave
 struFollow2.spaceStopSlave = 0 ; [°] slave stop space
 struFollow2.velPositSlave = 180 ; [°/s] Positioning speed
 struFollow2.accPositSlave = 360 ; [°/s^2] Positioning acceleration
 struFollow2.jerkPositSlave = 720 ; [ °/s^3] Jerk of positioning
 struFollow2.maxSpe = 1800 ; [°/s] Maximum slave velocity
 struFollow2.maxAcc = 18000 ; [°/s^2] Maximum slave acceleration
 struFollow2.kSlaveMaster = 360 / (k_duepi * slaveRadius) ; [°/mm] k
master -> slave conversion
endif
struFollow2.qMaster   = ip(1)
struFollow2.velMaster = iv(1)
struFollow2.accMaster = ia(1)
struFollow2.qSlave    = ip(2)
struFollow2.velSlave  = iv(2)
struFollow2.accSlave  = ia(2)
result = mv_follow2(struFollow2)
set_kinematics(ax2, struFollow2.qSlave, struFollow2.velSlave,
struFollow2.accSlave)
end_motion
end_rule
```

© 2025 Robox SpA

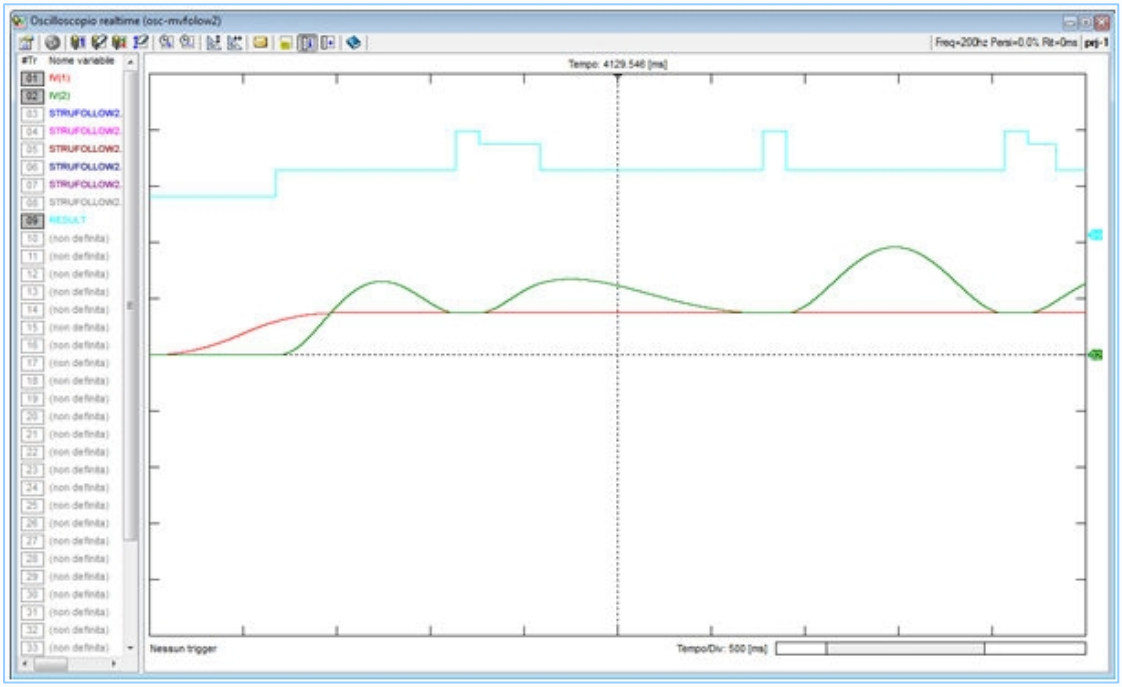


Example of use (variable cycle master and rotary slave):

```
stru_follow2 struFollow2

rule 1
motion
if(first_time())
   struFollow2.command.0 = 0 ; 1 = force slave stop at qParkSlave
 struFollow2.command.1 =  1 ; 1 = initial positioning to parking
quota
 struFollow2.command.2 = 0 ; 1 = initial positioning relative to
master loop
 struFollow2.command.16 = 1 ; 0 = fixed-cycle master, 1 = variable-
cycle master
 struFollow2.command.17 = 0 ; 1 = disable max_spe test
 struFollow2.command.18 = 0 ; 1 = disable max_acc test
 struFollow2.command.19 =  1 ; 1 = qSyncMaster aligned midway between
qSyncSlave and qOutSlave
 struFollow2.command.20 = 0 ; 1 = enable stop at qParkSlave (cyclic
slave)
 struFollow2.command.21 = 0 ; 1 = force slave movement forward only
(cyclic slave)
 struFollow2.command.22 =  1 ; 1 = enable on-the-fly sync recovery
 struFollow2.masterCycle = 300 ; [mm] Variable cycle master (300 mm
is the minimum cycle)
 struFollow2.spaceWaitInPark = 0 ; [mm] Master space with slave in
park.
 struFollow2.slaveCycle = 360 ; [°] Slave cycle
```

© 2025 Robox SpA

```
 struFollow2.qParkSlave = 180 ; [°] Slave parking quota
 struFollow2.qSyncSlave = -10 ; [°] Quota of  slave sync
 struFollow2.qQutSlave = 10 ; [°] Quota of  end sync slave
 struFollow2.spaceStopSlave = 0 ; [°] slave stop space
 struFollow2.velPositSlave = 180 ; [°/s] Positioning speed
 struFollow2.accPositSlave = 360 ; [°/s^2] Positioning acceleration
 struFollow2.jerkPositSlave = 720 ; [ °/s^3] Jerk of positioning
 struFollow2.maxSpe = 1800 ; [°/s] Maximum slave velocity
 struFollow2.maxAcc = 18000 ; [°/s^2] Maximum slave acceleration
 struFollow2.kSlaveMaster = 360 / (k_duepi * slaveRadius) ; [°/mm] k
master -> slave conversion
endif
struFollow2.qMaster   = ip(1)
struFollow2.velMaster = iv(1)
struFollow2.accMaster = ia(1)
struFollow2.qSlave    = ip(2)
struFollow2.velSlave  = iv(2)
struFollow2.accSlave  = ia(2)
; Master sync quota derived from an OB of type rc_elementlocation
struFollow2.qSyncMaster = ip(1) + buffer.get_distance_to_end(0)
result = mv_follow2(struFollow2)
set_kinematics(ax2, struFollow2.qSlave, struFollow2.velSlave,
struFollow2.accSlave)
end_motion
end_rule
```



Function mva_follow2()

This function is used to perform tracking of a SLAVE against a MASTER, with the master not associated with an axis. The function performs jerked movements computed with a pentic curve, cam of the position of the master axis.

Premise

Tracking can be single or cyclically repeated in fixed cycle or variable cycle.

The master could be a controlled physical axis, but also a simple external reference (e.g., a line encoder) or even a virtual quota constructed and evolved by the application software.

In any case, during the execution of the function, the master must always evolve in the same direction.

The slave could be a controlled physical axis, but also a simple unit constructed and evolved by the application software. It could be a linear axis of limited excursion or a cyclic axis (e.g., rotary).

Types of the follow2 function

The syntax of follow2 varies according to:

- MASTER and SLAVE both not associated with an axis -> mv_follow2 432()
- MASTER not associated with an axis, SLAVE associated with an axis -> mva_follow2()
- MASTER and SLAVE both associated with an axis (typical case) -> mva_follow2_m 443()

Functioning

The functioning follows the same concept explained in the mv_follow2 432() function.

Syntax	**i32 mva_follow2 (stru_follow2_s stru1 [, real period=0 [, stru_follow2info stru2=NULL])**
stru1	Structure of type stru_follow2_s 827 containing parameters related to movement
period	Time interval between two consecutive executions of this function. (optional, default 0) If 0 is assumed to be equal to SI
stru2	Structure of type stru_follow2info 827 where to save auxiliary data (out). (optional, default NULL)
Result	Variable containing information about the status of the function (see returns codes 61).

Value	Description
2 [M_POSITIONING]	Initial positioning requested by setting the relative bits of the *command* parameter
3 [M_WAITING]	The slave axis is waiting to start

© 2025 Robox SpA

Syntax	**i32 mva_follow2 (stru_follow2_s stru1 [, real period=0 [, stru_follow2info stru2=NULL])**
	<table><tr><th>Value</th><th>Description</th></tr><tr><td>4 [M_WAITING_NEXT_CYCLE]</td><td>No conditions to start, waiting for the next appointment</td></tr><tr><td>5 [M_FOLLOWING]</td><td>The slave axis has started tracking the master</td></tr><tr><td>8 [M_SYNCHRONIZED]</td><td>The slave axis has tracked and is synchronized to the master</td></tr><tr><td>17 [M_DECEL]</td><td>The slave axis has finished the synchronization phase and is decelerating</td></tr><tr><td>7 [M_HOMING]</td><td>The slave axis is returning to the parking point</td></tr><tr><td>-6 [M_IN_ALARM]</td><td>The function is in alarm, see *errorCode*</td></tr></table>
Validity	Rule
Note	The configuration of the function is done through the auxiliary data structure

Function mva_follow2_m()

This function is used to realize tracking of a SLAVE axis against a MASTER axis. The function performs jerked movements computed with a pentic curve, cam of the position of the master axis.

Premise

Tracking can be single or cyclically repeated in fixed cycle or variable cycle.

The master could be a controlled physical axis, but also a simple external reference (e.g., a line encoder) or even a virtual quota constructed and evolved by the application software.

In any case, during the execution of the function, the master must always evolve in the same direction.

The slave could be a controlled physical axis, but also a simple unit constructed and evolved by the application software. It could be a linear axis of limited excursion or a cyclic axis (e.g., rotary).

Types of the follow2 function

The syntax of follow2 varies according to:

- MASTER and SLAVE both not associated with an axis -> mv_follow2 432()

- MASTER not associated with an axis, SLAVE associated with an axis -> mva_follow2 442()
- MASTER and SLAVE both associated with an axis (typical case) -> mva_follow2_m()

Functioning

Functioning follows the same concept explained in the mv_follow2 432() function.

Syntax	**i32 mva_follow2_m (stru_follow2_sm stru1 [, real period=0 [, stru_follow2info stru2=NULL])**
stru1	Structure of type stru_follow2_sm 834 containing movement-related parameters
period	Time interval between two consecutive executions of this function. (optional, default 0) If 0 is assumed to be equal to SI
stru2	Structure of type stru_follow2info 827 where to save ancillary (out) data. (optional, default NULL)
Result	Variable containing information about the status of the function (see returns codes 61).

Value	Description
2 [M_POSITIONING]	Initial positioning requested by setting the relative bits of the *command* parameter
3 [M_WAITING]	The slave axis is waiting to start
4 [M_WAITING_NEXT_CYCLE]	No conditions to start, waiting for the next appointment
5 [M_FOLLOWING]	The slave axis has started tracking the master
8 [M_SYNCHRONIZED]	The slave axis has tracked and is synchronized to the master
17 [M_DECEL]	The slave axis has finished the synchronization phase and is decelerating
7 [M_HOMING]	The slave axis is returning to the parking point

© 2025 Robox SpA

Syntax	**i32 mva_follow2_m (stru_follow2_sm stru1 [, real period=0 [, stru_follow2info stru2=NULL])**
	Value: -6 [M_IN_ALARM] — Description: The function is in alarm, see *errorCode*
Validity	Rule
Note	The configuration of the function is done through the auxiliary data structure

Function mv_mot_exec()

Function that executes a previously created MOT (Multiple Output Table).

Functioning

During scanning (execution) the current key value*(key)* is entered and the corresponding interpolated values*(arr)* and additional data from the point beginning the interpolated segment*(stru)* are returned. The data are added to the maximum until filled and sorted in ascending order by *key*.

Syntax	**i32 mv_mot_exec (i32 motId, real key, real arr [, struct stru=NULL])**
motId	MOT reference code returned by mot_create 417()
key	Master coordinate (abscissa). This is the value of the current interpolation key. If a value less than or equal to the first is imposed, the execution index is set immediately at the beginning of the MOT. Similarly by imposing a value greater than or equal to the last, the execution index is set immediately at the end of the MOT
arr	Array of *n* data with the result of the MOT (ordered).
stru	Data structure of type struct 888 into which to download segment auxiliary data. (optional, default NULL)
Result	Returns: Value: 0 — Meaning: Operation ok Value: 1 — Meaning: Ok, underflow *(key* < smallest key)

© 2025 Robox SpA

Syntax	**i32 mv_mot_exec (i32 motId, real key, real arr [, struct stru=NULL])**

Value	Meaning
2	Ok, overflow *(key* > larger key)
-1	*motId* not existing
-3	*arr* illegal (smaller data size than expected)
-4	*stru* illegal (size not consistent with the size specified at creation)
-5	Internal error
-6	MOT manager busy (other transaction in progress)
-7	Empty (no points available)
-12	Interpolation type not managed

Validity	Rule
Note	

Examples

Example of use:

```
real posSlaveMot, dataFromMot, posMasterMod
; if you have more than a single data associated with the same master
position
; then dataFromMot will have to be an array of real

; rule
rule AUTO

    axes(ax_01)

    posMasterMod = abs_mod 134(ip(MASTER), 360.0)
    esitoMot = mv_mot_exec 445(rMotID, posMasterMod, dataFromMot)
    posSlaveMot = dataFromMot
    if(esitoMot = 0)
        ip(ax_01) = posSlaveMot
    end_if
```

```
end_rule
```

For a concrete example see in the RDE environment: Workspace -> Specials -> Examples Library.

Function mv_phase_adj()

Function to correct the phase position of a controlled axis (slave), which must move according to the position of a reference master (e.g., for controlling phasor belts).

HINT: We recommend using the more performant function mv_phase_adj2 448().

Functioning

The correction is made by adding (or subtracting) to the slave's base velocity a velocity triangle, calculated to recover (or lose) a space equal to the amount of phase error.

The velocity triangle acts in the period it takes the master to travel a specified space. The mission is calculated with a triangle possibly limited to the maximum velocity value.

Note: If speed and/or acceleration are limiting, the error value is not recovered all (but only the part that it can recover).

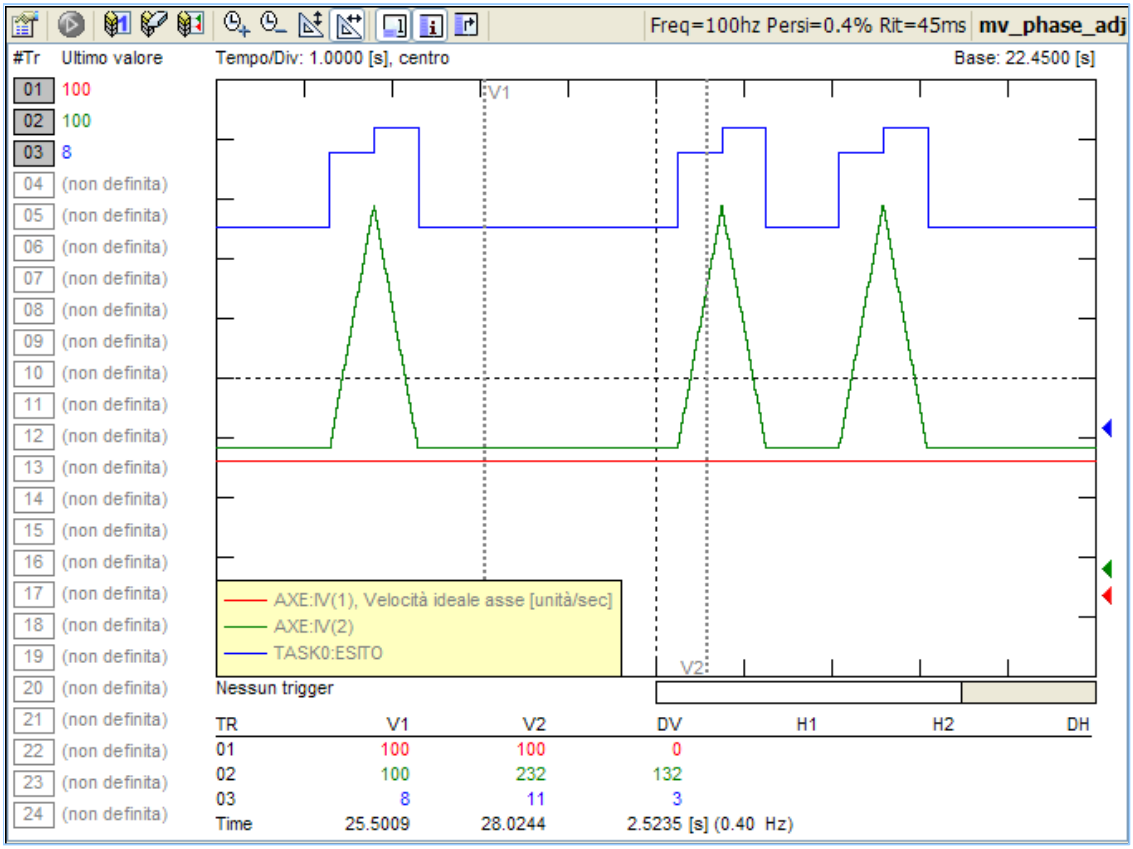
Syntax	**real mv_phase_adj (i32 state, stru_phadj stru)**
state	Variable containing information about the state of the function (see returns codes 61). <table><tr><th>Value</th><th>Description</th></tr><tr><td>11 [M_ADJUSTING_UP]</td><td>Correction in progress (triangle raising edge)</td></tr><tr><td>12 [M_ADJUSTING_DOWN]</td><td>Correction in progress (triangle falling edge)</td></tr><tr><td>8 [M_SYNCHRONIZED]</td><td>Movement completed (master has traveled the master space available for slave correction)</td></tr></table>
stru	Structure of type stru_phadj 868 containing parameters related to the movement
Result	Updated velocity value of the slave axis returned by the function
Validity	Rule
Note	The master axis can also evolve in the negative direction; the function provides for treating the incoming parameters *masterCycle* and *deltaMaster* as absolute values. The slave

© 2025 Robox SpA

Syntax	**real mv_phase_adj (i32 state, stru_phadj stru)**
	axis must always evolve in the positive direction, and its velocity *vRef* must be positive

Examples

Example on oscilloscope of a possible output of the function:



For a concrete example see in the RDE environment: Workspace -> Specials -> Examples Library.

Function mv_phase_adj2()

Function to correct the phase position of a controlled axis (slave) that has to move according to the position of a reference master (e.g., for controlling phasing belts).

Functioning

To start correction, the error must be set and then the state variable must be assigned the value M_INITIALIZING (at least the first time) or, for subsequent corrections, restart from 0 with the master position.

The correction is made by adding (or subtracting) to the slave's base speed a correction speed calculated to recover (or lose) a space equal to the amount of the error.

© 2025 Robox SpA

By the *corrType* parameter, it is possible to choose from several types of correction speed profiles:

- 0, triangular
- 1, parabolic
- 2, sinusoidal
- 3, jerky (parabolic with continuous acceleration)
- 4, raised cosine
- 5, pentastic

When initializing the correction, the peak velocity of the correction is calculated using the master's reference velocity*(vRefMaster*), and if it exceeds one of the limiting velocities*(vMin* or *vMax*) the correction triangle (or equivalent) is automatically transformed into a trapezoid, increasing the initial and final acceleration until *aMax* is reached.

The correction acts in the period it takes the master to travel a specified space.

The slave's speed and acceleration are eventually limited to the values of minimum and maximum speed and maximum acceleration by signaling in the state variable that this has occurred.

If speed and/or acceleration are limiting, the error value is not recovered all.

Syntax	**real mv_phase_adj2 (i32 state, stru_phadj2 stru)**
state	Variable containing information about the state of the function (see returns codes 61).

Value	Description
0	Function not active, slave copies the reference speed
1 [M_INITIALIZING]	Start of correction (set by the user)
11 [M_ADJUSTING_UP]	Correction in progress (triangle raising edge)
12 [M_ADJUSTING_DOWN]	Correction in progress (triangle falling edge)
8 [M_SYNCHRONIZED]	Movement completed (master has traveled the master space available for slave correction)
10 [M_SPEED_LIMITED]	Speed has been limited to minimum or maximum values (correction will be partial)

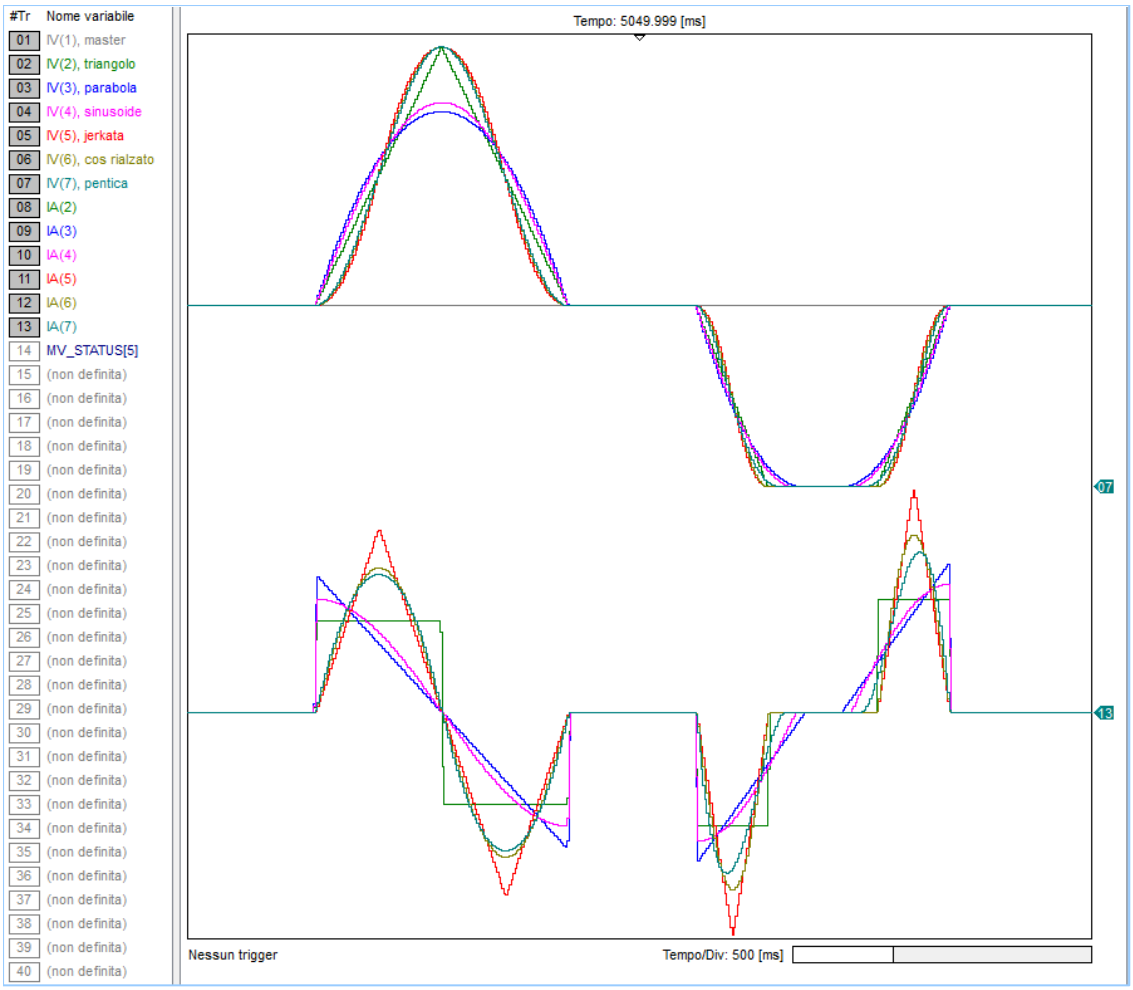
© 2025 Robox SpA

Syntax	**real mv_phase_adj2 (i32 state, stru_phadj2 stru)**
	Value: 9 [M_ACCEL_LIMITED] — Description: Speed has been limited to meet the maximum acceleration value (correction will be partial)
stru	Structure of type stru_phadj2 871 containing motion-related parameters
Result	Updated velocity value of the slave axis returned by the function
Validity	Rule
Note	The master axis must always evolve in the positive direction, not necessarily at constant speed

Examples

Example on oscilloscope of a possible output of the function:

© 2025 Robox SpA



For a concrete example see in the RDE environment: Workspace -> Specials -> Examples Library.

Function mv_ramp()

WARNING: Deprecated. Use ramp() function. 530

Function mv_reach_target()

Function that calculates the velocity to move the slave axis by a specified space and in a specified time.

Functioning

It calculates the velocity to move the slave axis of the given space (parameter *ds*) in the given time (parameter *tm*), moving from the current velocity *velSlave* of the axis to the synchronization velocity *vTarget* (respecting the parameters *velMinSlave*, *velMaxSlave*, *accMaxSlave*), with acceleration (calculated automatically) constant and as little as possible.

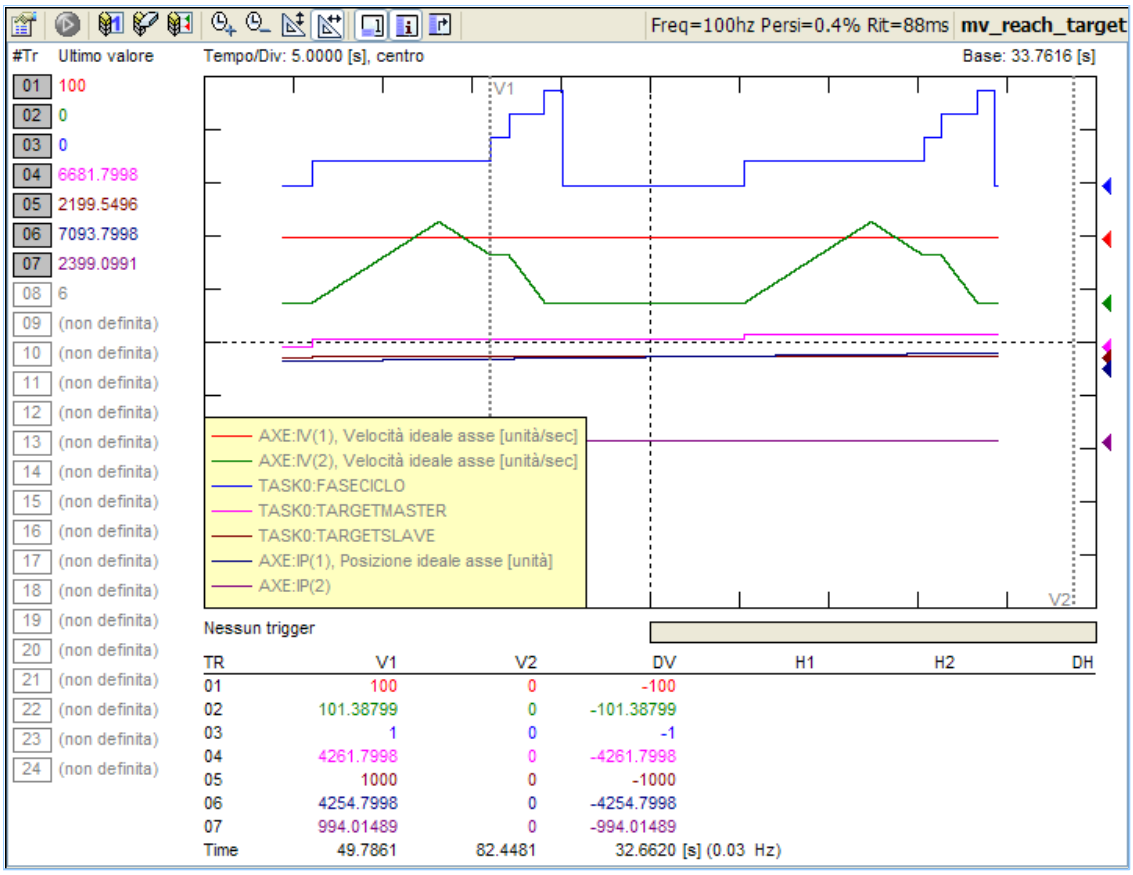
© 2025 Robox SpA

If *tm<=0* or *ds<=0* we move as fast as possible toward *vTarget*, using *accMaxSlave* as acceleration.

Syntax	**real mv_reach_target (i32 return, stru_reachtarget stru)**
return	Variable containing information about the status of the function (cedi returns codes 61). Value / Description -4 [M_LOW_ACCERATION] — *accMaxSlave* = 0 5 [M_FOLLOWING] — Movement in progress 6 [M_REACHED] — Movement completed
stru	Structure of type stru_reachtarget 876 containing parameters related to the movement
Result	Updated value of slave axis speed returned by the function
Validity	Rule
Note	The tracking movement of the slave axis must always be in the positive direction(*ds* > 0)

Examples

Example on oscilloscope of a possible output of the function:



For a concrete example see in the RDE environment: Workspace -> Specials -> Examples Library.

Function mv_sine_cam()

Function that executes a cam (y as a function of x) with sinusoidal y speed.

Syntax	**real mv_sine_cam (real xCurr, real xCycleLen, real yCycleLen)**
xCurr	Current value of x (current master space). Can take values between 0 and *xCycleLen*
xCycleLen	Length of the cycle in x (master space)
yCycleLen	Length of the cycle in y (slave space)
Result	Position of y calculated from *xCurr*
Validity	Rule
Note	

Examples

Example of use:

```
; movement of a slave axis to always rotate in the same direction
; depending on the progress of the master
select(fase)
case 0
    ; initialization
    initMaster = ip(ax_master)
    initSlave = ip(ax_slave)
    fase = 1
    ; break voluntarily omitted
case 1
    ; raising
    xCur = ip(ax_master)-initMaster
    if(xCur > masterCycle)
        initMaster = initMaster + masterCycle
        initSlave = initSlave  + masterSlave
        xCur = xcur - masterCycle
    endif
    ip(slave) = initSlave + mv_sine_cam(xCur, masterCycle,
masterSlave)
    break
endselect
```

Function mv_synchro()

Function to synchronize a slave axis to a master axis such that the slave reaches the synchronism quota, with specified speed, at each cycle of the master. The synchronism occurs for a single point.

Functioning

The function moves the specified slave unit from the current *qSlave* position to a *qSyncSlave* destination, meeting the appointment with a master moving at *velMaster* speed to its appointment at the *qSyncMaster* quota.

The movement occurs with accelerations and velocities recalculated in each call of the function.

In the event that the slave's velocity exceeds *velMaxSlave* or acceleration exceeds *accMaxSlave,* they are restricted and signaled via the function return code.

The quota *qSyncSlave* is reached by the slave with the *velSyncSlave* speed.

Note: *velSyncSlave* usually coincides with *velMaster*, but not necessarily.

The *velMaster* can change, so the calculations are repeated at each sampling and are only used to decide the opportunity in that sampling to accelerate/decelerate the slave.

The *velMaster* is taken as an absolute value and is always understood to agree with the speed at which the slave approaches the *qSyncSlave* appointment.

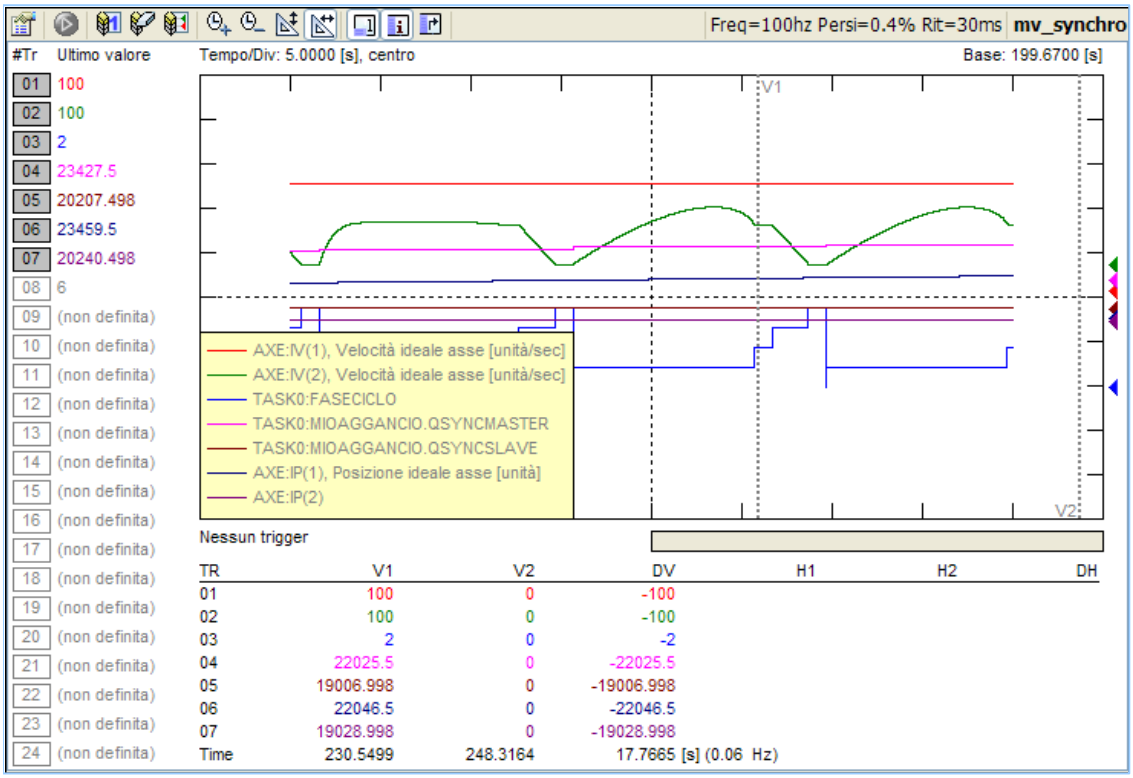
kappa allows the trajectory of the slave to be warped: the higher the value, the more the slave movement is anticipated, i.e., more space is attempted during the first half of the mission. Reasonable values range from 1.0 (no advance) up to 3.0 (maximum advance).

© 2025 Robox SpA

Syntax	**real mv_synchro (i32 return, stru_synchro stru)**
return	Variable containing information about the status of the function (returns codes 61)
stru	Structure of type stru_synchro 884 containing parameters related to the handling
Result	Updated position value of the slave axis returned by the function
Validity	Rule
Note	

Examples

Example on oscilloscope of a possible output of the function:



For a concrete example see in the RDE environment: Workspace -> Specials -> Examples Library.

Function mv_table()

Function that allows a point cam to be performed.

Functioning

© 2025 Robox SpA

Depending on an input unit (master), the corresponding value of the slave unit is calculated using a table of points (the result, for example, of spreadsheets or mathematical programs) that has been previously loaded onto the CF of the control(f_read_table_s 308()).

The interpolation used between the points is of the linear type.

By optional parameters it is possible to perform the transition between two cams or the weighted sum of it.

Syntax	**real mv_table (real xCurr, i32 nPoints1, stru_cam stru1 [, real par=0] [, i32 nPoints2=0] [, stru_cam stru2=NULL])**
xCurr	Master coordinate (abscissa)
nPoints1	Number of points of which the first cam is composed (x-y value pairs). Must match the size of the array *stru1*
stru1	Array of structures of type stru_cam 804 containing the values of the first cam (length *nPoints1*)
par	Mixer parameter that allows switching between the two cams. (optional, default 0) Value / Meaning 0 / Executes cam 1 0 -:- 1 (extremes not included). / Executes a weighted overlap of the two cams 1 / Executes cam 2
nPoints2	Number of points of which the second cam is composed. (optional, default 0) Must match the size of the array *stru2*
stru2	Array of structures of type stru_cam containing the values of the second cam (length *nPoints2*). (optional, default NULL)
Result	Updated value of the y-coordinate
Validity	Rule
Note	

Diagnostics

- Generates 9900 ill. Arg A:5 T:x ST:xxx if *par<>0* and *nPoints2=0*.

Examples

© 2025 Robox SpA

Example of use:

```
; tab1 variable definition of type stru_cam
stru_cam 804 tab1[7]
if(first_time 321())
    tab1[0].x = 0
    tab1[1].x = 90
    tab1[2].x = 135
    tab1[3].x = 180
    tab1[4].x = 270
    tab1[5].x = 330
    tab1[6].x = 360
    tab1[0].y = 10
    tab1[1].y = 90
    tab1[2].y = 250
    tab1[3].y = 300
    tab1[4].y = 270
    tab1[5].y = 100
    tab1[6].y = 10
    rr(100) = 1
endif
ip(1) = mod 412(rr(100) * tfb * 360.0, 360)
ip(2) = mv_table 455(ip(1), 7, tab1)
```

Function mv_to()

Function that brings the indicated unit to the set value with a trapezoidal velocity profile.

NOTE: It is analogous to the mv_to 459() function of the same name with inline parameterization.

Syntax	**real mv_to (i32 result, stru_mvto stru [, real overDec=0])**
return	Variable containing information about the state of the function (see returns codes 61).

Value	Description
14 [M_HOLDED]	Desired speed or feedrate set to 0
6 [M_REACHED]	Target reached
15 [M_ACCEL]	Acceleration in progress
13 [M_OVERSHOOTING]	Target trespassed
17 [M_DECEL]	Deceleration in progress
16 [M_AT_SPEED]	Movement at speed

© 2025 Robox SpA

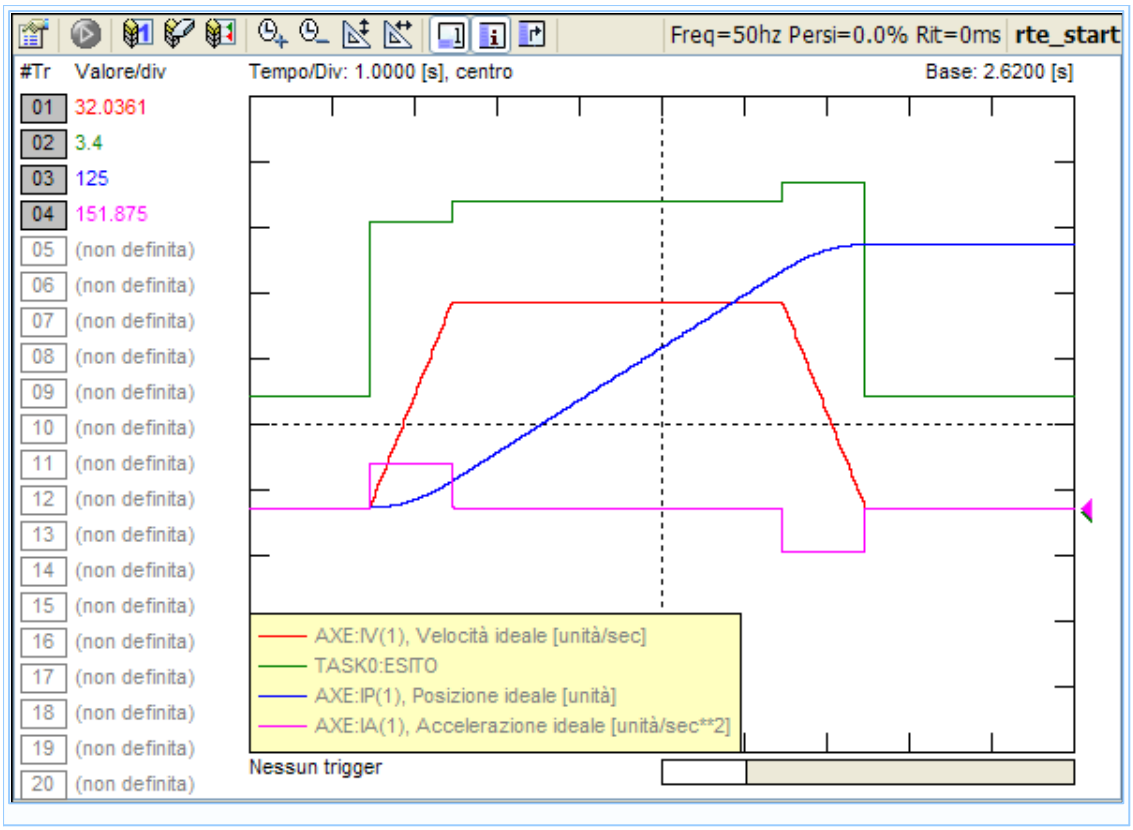
Syntax	**real mv_to (i32 result, stru_mvto stru [, real overDec=0])**
stru	Structure of type stru_mvto 853 containing parameters related to movement
overDec	Deceleration percentage to be applied in case the mission target is approached runtime when it is already decelerating. (optional, default 0) If negative, 0 is assumed. The resulting acceleration will be: *dec +(overDec* / 100) * *dec.* (e.g. if *dec* = 120 and *overDec* = 25 the deceleration will result 150)
Result	Updated value returned by the function
Validity	Rule
Note	*overDec* can be useful in case of calculation of stop target, on variation of digital input (e.g., photocell)

Examples

Example of use:

```
; local definitions
stru_mvto 853 myMov ; variable of type stru_mvto
int result

; rule
; current position [units]
myMov.cPos = ip(AX_1)
; current speed [units/sec]
myMov.cSpeed = iv(AX_1)
; desired position [units]
myMov.dPos = rr(2000)
; desired speed [units/sec]
myMov.dSpeed = rr(2001)
; acceleration [units/sec²]
myMov.acc = rr(2002)
; deceleration [units/sec²]
myMov.dec = rr(2003)
; value of SI
myMov.delta = SI
ip(AX_1) = mv_to 457(result, myMov, overdec)
; (vel 100, acc 100, dec 100, space 500)
```

© 2025 Robox SpA



Function mv_to(), parameterized inline

Function that brings the specified axis to the predetermined position value with a trapezoidal velocity-time profile.

NOTE: It is analogous to the mv_to 457() function of the same name with parameterization by structure.

Syntax	**real mv_to (i32 result, i32 nAx, real pDes, real sDes, real aDes [, real dDes=-aDes] [, real sovraDec=0])**
result	Variable containing information about the status of the function (see returns codes 61).

Value	Description
14 [M_HOLDED] (if *overDec* exists = 0)	Desired speed or feedrate set to 0
6 [M_REACHED]	Target reached
15 [M_ACCEL]	Acceleration in progress
13 [M_OVERSHOOTING]	Target trespassed

Syntax	**real mv_to (i32 result, i32 nAx, real pDes, real sDes, real aDes [, real dDes=-aDes] [, real sovraDec=0])**
	Value / Description 17 [M_DECEL] / Deceleration in progress 16 [M_AT_SPEED] / Moving at speed -5 [M_ILLEGAL_ARGS] / Incorrect axis
nAx	Number of the axis to be moved (1-32)
pDes	Target position
sDes	Speed
aDes	Acceleration
dDes	Deceleration. (optional, default *-aDes*)
sovraDec	Deceleration percentage to be applied in case the mission target is approached runtime when it is already decelerating. (optional, default 0) If negative, 0 is assumed. The resulting acceleration will be: *dDes +(overDec* / 100) * *dDes.* (e.g. if *dDes* = 120 and *overDec* = 25 the deceleration will result 150)
Result	Updated value returned by the function
Validity	Rule
Note	*overDec* can be useful in the case of stop target calculation, on variation of a digital input (e.g., photocell). If CAM alarm enabled the mission is limited to end of strokes. The FR is not honored

Examples

Example of use:

```
ip(1) = mv_to(esito, 1, rr(2000), rr(2001), rr(2002))
; in rr(2000) is contained the target position value
; in rr(2001) is contained the value of velocity
; in rr(2002) is contained the value of acceleration
```

For a concrete example see in the RDE environment: Workspace -> Specials -> Examples Library.

© 2025 Robox SpA

Function mv_to_cj()

Function that performs movement to a target (with a stationary start) of a generic unit by specifying desired velocity, acceleration and jerk.

The parameters of desired velocity, acceleration and jerk are entered into the relative structure.

The data are assumed only on *start*. On start with velocity other than 0, there is a stop phase of the unit before starting toward the programmed target.

Hold and fr changes are handled using a linear velocity ramp.

NOTE: It is analogous to the function mv_to_cjv 461() but the latter allows starting from initial velocity other than 0 and changing runtime parameters.

Syntax	**real mv_to_cj (i32 return, i32 start, stru_mvtocj stru [, i32 holdFlag=0] [, real fr=1])**
return	Variable containing information about the state of the function (see returns codes 61).
start	Variable used to have the mission parameters acquired (request with any value <> 0). The function resets the request by writing 0 to it
stru	Variable of type stru_mvtocj 855 containing the function parameters
holdFlag	Flag to command the hold state to the mission. (optional, default 0)
fr	Velocity reduction value (0-1). (optional, default 1) See fr
Result	Unit position value
Validity	Rule
Note	

For a concrete example see in RDE environment: Workspace -> Specials -> Examples library.

Function mv_to_cjv()

Function that brings the controlled unit from the current speed (corresponding to the first function call) to the desired speed, controlling acceleration with the specified jerk.

Variation of the desired units (target position, steady state speed, acceleration and jerk) is possible at any time.

The function ends its work when it stops at the specified target position.

Functioning

The current data*(cPos*, *cSpeed*, *cAcc*) typically are imposed only when the first call to the function is made, after which they are updated by the function.

© 2025 Robox SpA

The returned values of *cPos*, *cSpeed* and *cAcc*, in case this function is used to drive an axis, must be set via the set_kinematics 569() instruction.

The desired data*(dPos*, *dSpeed*, *dAcc*, *dJerk*) can be changed at any time.

NOTE: It is analogous to the mv_to_cj 461() function, but this one only allows you to start with an initial velocity of 0.

Syntax	**i32 mv_to_cjv (stru_mvtocjv| stru_mvtocjve stru [, i32 holdFlag=0] [, real fr=1])**
stru	Variable of type stru_mvtocjv 857 or stru_mvtocjve 861 containing the parameters of the function
holdFlag	Flag to command the hold state to the mission. (optional, default 0)
fr	Velocity reduction value (0-1). (optional, default 1) See fr
Result	Returns the state of the function (see return codes 61): Value / Description -6 [M_IN_ALARM] / Alarm present 2 [M_POSITIONING] / Initial positioning 13 [M_OVERSHOOTING] / Target position exceeded 15 [M_ACCEL] / Acceleration in progress 16 [M_AT_SPEED] / Movement at speed 17 [M_DECEL] / Deceleration in progress 6 [M_REACHED] / Target reached
Validity	Rule
Note	In the stru_mvtocjve, in case of *dSpeed=0*, you have automatic writing of the stop position and its time required. It can generate alarm 9900 (see configuration flag sys_flag.4). You can increase the diagnostics in the report via the predefined variable sys_flag.10

Examples

© 2025 Robox SpA

Example of use:

```
if(first_time 321())
    ; code executed just the fist time
    ; set parameter of the structure
    ; sample interval
    mv.delta = SI
    ; actual posiz [units]
    mv.cPos = IP(AX_1)
    ; actual speed [units/sec]
    mv.cSpeed = IV(AX_1)
    ; actual acc [units/sec*sec]
    mv.cAcc = IA(AX_1)
endif

if(rRichiestaStop)
    rRichiestaStop = FALSE
    HOLD = TRUE
endif

; desired posiz [units]
mv.dpos = rrDesPos
; desired speed [units/sec]
mv.dSpeed = rrDesSpeed
; desired acc [units/sec*sec]
mv.dacc = rrDesAcc
; desired jerk [units/sec*sec*sec]
mv.djerk = rrDesJerk
; call the function
esito = mv_to_cjv 461(mv, HOLD, FR)
; link to the axe
set_kinematics 569(AX_1, mv.cPos, mv.cSpeed, mv.cAcc)
```

Function mv_to_cjv_info()

Function that calculates the space and time required to reach the desired speed with specified acceleration and jerk.

Current*(cPos*, *cSpeed*, *cAcc*) and desired*(dAcc*, *dJerk*, *targetSpeed*) kinematics data must be entered. The *posInfo and timeInfo* parameters are then returned, i.e., the position at which the desired target speed will be reached and the time required to reach it.

Syntax	**i32 mv_to_cjv_info (stru_mvtocjv_info stru)**
stru	Variable of type stru_mvtocjv_info 859 containing the parameters of the function
Result	Returns a TRUE value if all parameters are correct, otherwise if acceleration or jerk is equal to 0 returns FALSE and alarm 9900 illegal arg is generated

© 2025 Robox SpA

Syntax	**i32 mv_to_cjv_info (stru_mvtocjv_info stru)**
Validity	Rule, Task
Note	Typically this function is used in conjunction with mv_to_cjv 461(). In that case, it is essential to provide the same input data as current values

Function mv_to_vel()

Function that performs the movement of a generic unit, moving initially at a certain speed, to a target point to be reached with desired cruising speed and final speed.

Syntax	**real mv_to_vel (i32 return, real pAct, real vAct, real pDes, real vDes, real vFin, real baseTime, real acc [, real dec=acc] [, real perc=0.8])**
return	Variable containing information about the status of the function (see returns codes 61).

Value	Description
-5 [M_ILLEGAL_ARGS]	Illegal parameter(*acc=0* or *baseTime=0*)
13 [M_OVERSHOOTING]	Target exceeded. The move proceeds inertially
-4 [M_LOW_ACCELERATION]	Too fast to reach target point with specified *vFin*
6 [M_REACHED]	Target point reached (applies only if you have specified *vFin=0*)
15 [M_ACCEL]	Acceleration in progress
16 [M_AT_SPEED]	Motion at speed with *vDes*
17 [M_DECEL]	Deceleration in progress
18 [M_LAST_CYCLE]	Last cycle before passing target

pAct	Current position
vAct	Current speed
pDes	Position to be reached

© 2025 Robox SpA

Syntax	**real mv_to_vel (i32 return, real pAct, real vAct, real pDes, real vDes, real vFin, real baseTime, real acc [, real dec=acc] [, real perc=0.8])**
vDes	Desired speed (intended in absolute value)
vFin	Final speed (intended in absolute value)
baseTime	Time base (normally SI)
acc	Desired acceleration (intended in absolute value)
dec	Desired deceleration (intended in absolute value). (optional, default *acc*)
perc	Percentage value affecting deceleration (0-1). (optional, default 0.8) During arrival calculations at the target point, the number of samplings required to arrive from the current position to the desired position with the specified velocity *vFin* is evaluated. This number is truncated to the lower integer if the remainder is < *perc* (so the actual deceleration will be a little larger than *dec*) or will be increased to the next integer if the remainder is > *perc* (lower resultant deceleration)
Result	Updated position value
Validity	Rule
Note	*vFin* is understood to agree with the direction of the requested movement (if *pDes* > *pAct*, *vFin* is understood to be positive, otherwise it is understood to be negative). Updating the velocity *vAct* must be done externally to the function by the user

If *vFin* > 0, the target position is passed, after which the function continues inertially as long as it is invoked. If *vFin* = 0, the function stops at the desired point.

If the deceleration parameter does not allow it to get from *pAct* to *pDes* with the specified *vFin*, mv_to_vel() still tends to reach *vFin*; *pDes* will be passed with a speed other than *vFin*, and the movement will continue inertially with speed tending progressively toward *vFin*.

NOTE: The function can also be called with the axis already in motion.

Examples

Example of use:

```
select(fase)
; ...
case 4 ; I go to P4 with final speed V2 and cruising speed V2
    ip(asse_4) = mv_to_vel(esito, ip(asse_4), iv(asse_4), P4, V2,
V2, SI, targetAcc, targetDec, 0.5)
```

```
    if(esito = M_OVERSHOOTING)
        fase = 5
    endif
    break
case 5 ; I stop on P5 with targetDec
    ip(asse_4) = mv_to_vel(esito, ip(asse_4), iv(asse_4), P5, V2, 0,
SI, targetAcc, targetDec, 0.5)
    if(esito = M_REACHED)
        tm_on(t1, TRUE)
        fase = 6
    endif
    break
; ...
endselect
```

Function mv_tracking()

WARNING: Deprecated. Use tracking() function.

Function mva_jog()

Function that performs axis movement in manual mode.

Compared with mva_jog2():

- It is not possible to distinctly set acceleration and deceleration
- It is not possible to set acceleration and deceleration jerk distinctly
- Cannot handle stop on end of strokes with jerked motion

Syntax	**i32 mva_jog (i32 nAx, i32 dir, real spe, real acc [, real jer=1e100] [, i32 noEEOS=0])**
nAx	Number of the axis (1-:-32)
dir	Direction: Value: Meaning -1: Negative direction 0: Stationary axis 1: Positive direction
spe	Speed
acc	Acceleration
jer	Jerk. (optional, default 1e100)
noEEOS	Flag to disable the use of EEOS (Electronic End Of Stroke) as an end of stroke: (optional, default 0)

© 2025 Robox SpA

Syntax	**i32 mva_jog (i32 nAx, i32 dir, real spe, real acc [, real jer=1e100] [, i32 noEEOS=0])**
	<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>EEOS enabled</td></tr><tr><td>1</td><td>Disabled</td></tr></table> If disabled, CRASH alarms are not handled, even if CAM enabled. **NOTE**: See predefined variables min_str and max_str.
Result	Variable containing information about the status of the function (see returns codes 61). <table><tr><th>Value</th><th>Description</th></tr><tr><td>-5 [M_ILLEGAL_ARGS]</td><td>Illegal parameter (*acc=0* or *nAx* illegal)</td></tr><tr><td>-6 [M_IN_ALARM]</td><td>Alarm (ip axis already set)</td></tr><tr><td>6 [M_REACHED]</td><td>Target reached. This means that either the direction is 0 and the axis is stationary, or the direction is +1 or -1 and the axis has reached one of the end of strokes</td></tr><tr><td>15 [M_ACCEL]</td><td>Acceleration in progress</td></tr><tr><td>16 [M_AT_SPEED]</td><td>Movement at speed</td></tr><tr><td>17 [M_DECEL]</td><td>Deceleration in progress</td></tr></table>
Validity	Rule
Note	Acting on the jerk results in an 'S' velocity profile, thus a smoother motion. Stopping on end of strokes is done with maximum jerk

Diagnostics

- Generate alarm 9900 ill. Arg. A:2 T:x ST:xxx if *nAx* invalid
- Generate alarm 9900 ill. Arg. A:4 T:x ST:xxx if *acc* = 0
- Generate alarm 2800 Double Position Set T:x ST:xxx if already set axis ip

© 2025 Robox SpA

Examples

Example of use:

```
rule 100
axes 1
motion
    esito = mva_jog(1, jog, speed, accel, accel*10, 1)
end_motion
end_rule
```

For a concrete example see in the RDE environment: Workspace -> Specials -> Examples Library.

Function mva_jog2()

Function that performs axis movement in manual mode.

Compared with mva_jog() it allows you to:

- Set distinctly acceleration and deceleration
- Distinctly set jerk on acceleration and deceleration
- Manage stop on end of strokes with jerked motion

Syntax	**i32 mva_jog2 (i32 nAx, i32 dir, real spe, real acc, real dec [, real jerAcc=0] [, real jerDec=0] [, i32 noEEOS=0])**
nAx	Axis number (1-:-32)
dir	Direction: Value: Meaning -1: Negative direction 0: Stationary axis 1: Positive direction
spe	Speed
acc	Acceleration
dec	Deceleration
jerAcc	Acceleration jerk (0 no jerk, 1 all acceleration phase with controlled jerk). (optional, default 0)
jerDec	Deceleration jerk (0 no jerk, 1 all deceleration phase with controlled jerk). (optional, default 0)
noEEOS	Flag to disable the use of EEOS (Electronic End Of Stroke) as an end of stroke: (optional,

© 2025 Robox SpA

Syntax	**i32 mva_jog2 (i32 nAx, i32 dir, real spe, real acc, real dec [, real jerAcc=0] [, real jerDec=0] [, i32 noEEOS=0])**
	default 0) Value / Meaning 0 / EEOS enabled 1 / Disabled If disabled, CRASH alarms are not handled, even if CAM enabled. **NOTE**: See predefined variables min_str and max_str.
Result	Variable containing information about the status of the function (see returns codes 61). Value / Description -5 [M_ILLEGAL_ARGS] / Illegal parameter(*acc=0* or *nAx* illegal) -6 [M_IN_ALARM] / Alarm (ip axis already set) 6 [M_REACHED] / Target reached. This means that either the direction is 0 and the axis is stationary, or the direction is +1 or -1 and the axis has reached one of the end of strokes 15 [M_ACCEL] / Acceleration in progress 16 [M_AT_SPEED] / Movement at speed 17 [M_DECEL] / Deceleration in progress
Validity	Rule
Note	Acting on the jerk results in an 'S' velocity profile, thus a smoother motion

Examples

Example of use:

```
; movement with equal acceleration and deceleration and maximum jerk
effect
```

© 2025 Robox SpA

```
rule 100
axes 1
motion
    esito = mva_jog2(1, jog, 10, 100, 100, 1, 1, 0)
end_motion
end_rule

; movement with smoother acceleration than deceleration
; and with the jerk effect greater in acceleration
rule 100
axes 1
motion
    esito = mva_jog2(1, jog, 10, 100, 200, 1, 0.5, 1)
end_motion
end_rule

rule 100
axes 1
motion
    esito = mva_jog2(1, jog, 10, 100,100) ; like mva_jog
end_motion
end_rule
```

For a concrete example see in the RDE environment: Workspace -> Specials -> Examples Library.

Function mva_open_loop()

Function to be used in missing_power rules. It sets:

- IP = CP (the quota read from the transducer is copied to Ideal Position).
- Does not generate CRASH alarm even if enabled
- Sets IV and IA to 0.

From RTE version 34.27.02, as soon as the axis is enabled the ideal velocity is set to 0.

NOTE: In case you want to use the old behavior, set sys_cfg_2.12 to 1.

Syntax	**i32 mva_open_loop (i32 nAx)**
nAx	Number of the axis (1-:-32)
Result	0 if Ok, otherwise error
Validity	Rule
Note	

Note: In case Planar Table Compensation via sys_cfg.12 is enabled, IP(ax) = CP_C(ax) is imposed.

Examples

Example of use:

```
rule 100
axes 1
motion
    esito = mva_open_loop(1)
end_motion
end_rule
```

Function mva_to_n()

Function that moves the given axes (1-:-32) toward the target position value set with a velocity profile toward trapezoidal time.

Note: The difference with the mva_to_n_v() function is that it cannot be called with axis(s) already in motion and the final velocity on the target cannot be specified.

Syntax	**i32 mva_to_n (i32 start, stru_mvitp stru [, i32 nDef=1] [, i32 holdFlag=0] [, real fr=1])**
start	Flag to enable data re-acknowledgement (only if axes stopped). The flag is reset when the data is taken
stru	Structure of type stru_mvitp containing motion-related parameters
nDef	Array size (number of axes involved). (optional, default 1)
holdFlag	Flag to command HOLD status to the mission. (optional, default 0)
fr	Speed reduction value (0-1). (optional, default 1) See fr
Result	Variable containing information about the status of the function (see returns codes).

Value	Description
-5 [M_ILLEGAL_ARGS]	Illegal parameter (*acc=0* or illegal axis number)
-6 [M_IN_ALARM]	Alarm (axis ip already set)
2 [M_POSITIONING]	Initial positioning
6 [M_REACHED]	Target reached. This means that either the direction is 0 and the axis is stationary,

© 2025 Robox SpA

Syntax	**i32 mva_to_n (i32 start, stru_mvitp stru [, i32 nDef=1] [, i32 holdFlag=0] [, real fr=1])**
	<table><tr><th>Value</th><th>Description</th></tr><tr><td></td><td>or the direction is +1 or -1 and the axis has reached one of the end of strokes</td></tr><tr><td>14 [M_HOLDED]</td><td>Motion stopped due to HOLD flag. It is also returned in cases where the desired axis speed is imposed to 0 or the feedrate is set to zero</td></tr><tr><td>15 [M_ACCEL]</td><td>Acceleration in progress</td></tr><tr><td>16 [M_AT_SPEED]</td><td>Motion at speed</td></tr><tr><td>17 [M_DECEL]</td><td>Deceleration in progress</td></tr></table>
Validity	Rule
Note	The mission may involve multiple axes (max 32). They will be linearly interpolated in order to reach their targets all together

Diagnostics

- Generate alarm 9900 ill. Arg. A:2 T:x ST:xxx if invalid axis number
- Generates alarm 9900 ill. Arg. A:3 T:x ST:xxx if *acc* = 0
- Generate alarm 9903 Par. are changed T:x ST:xxx if running axis sequence changed
- Generate alarm 2800 Double Position Set T:x ST:xxx if already imposed axis ip

Examples

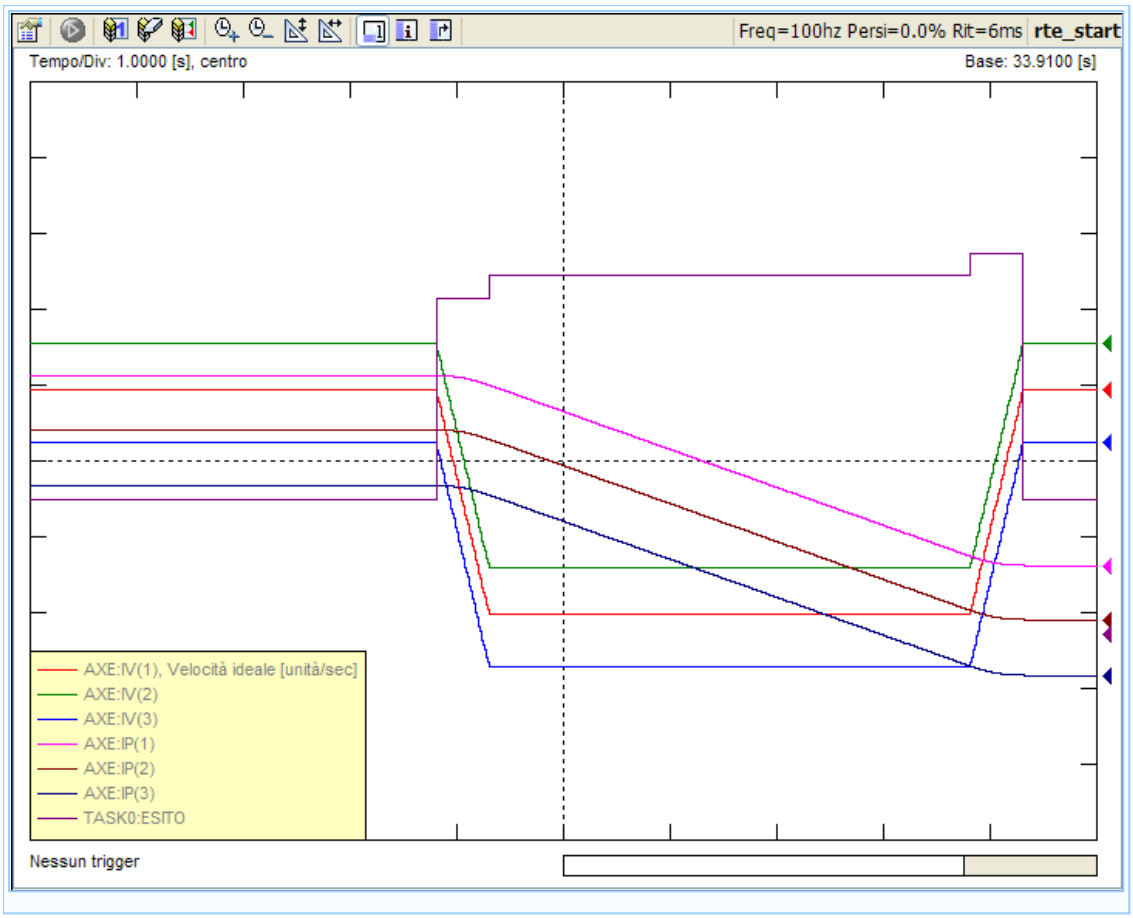
Example of use:

```
; example of three axes moving interpolated back and forth between
two points.
; definitions
stru_mvitp 850 m1[3]
int restartFlag
int holdflg

motion
if(first_time() 321)
```

```
    rr(1000) = 100 ; target ax 1
    rr(1001) = 10  ; vel
    rr(1002) = 20  ; acc
    rr(1003) = 20  ; dec
    rr(2000) = 100 ; target ax 2
    rr(2001) = 10  ; vel
    rr(2002) = 20  ; acc
    rr(2003) = 20  ; dec
    rr(3000) = 100 ; target ax 3
    rr(3001) = 10  ; vel
    rr(3002) = 20  ; acc
    rr(3003) = 20  ; dec
    restartFlag = true
endif

; AXIS 1
;*********
; axis number
m1[0].nAx = 1
; desired position [units]
m1[0].target = rr(1000)
; desired speed [units/sec]
m1[0].v = rr(1001)
; acceleration [units/sec²]
m1[0].acc = rr(1002)
; deceleration [units/sec²]
m1[0].dec = rr(1003)
; dec hold value
m1[0].decH = max_acc(1)
; AXIS 2
;*********
; axis number
m1[1].nAx = 2
; desired position [units]
m1[1].target = rr(2000)
; desired speed [units/sec]
m1[1].v = rr(2001)
; acceleration [units/sec²]
m1[1].acc = rr(2002)
; deceleration [units/sec²]
m1[1].dec = rr(2003)
; dec hold value
m1[1].decH = max_acc(2)
; AXIS 3
;********
; axis number
m1[2].nAx = 3
; desired position [units]
m1[2].target = rr(3000)
; desired speed [units/sec]
```

© 2025 Robox SpA

```
m1[2].v = rr(3001)
; acceleration [units/sec²]
m1[2].acc = rr(3002)
; deceleration [units/sec²]
m1[2].dec = rr(3003)
; dec hold value
m1[2].decH = max_acc(3)
esito = mva_to_n(restartFlag, m1[0], 3, hold, fr)
if(rise(esito = M_REACHED))
    if(avanti)
        inc(rr(1000),100)
        inc(rr(2000),100)
        inc(rr(3000),100)
        avanti = false
        restartFlag = true
    else
        dec(rr(1000),100)
        dec(rr(2000),100)
        dec(rr(3000),100)
        avanti = true
        restartFlag = true
    endif
endif
end_motion
```



Function mva_to_n_cj()

Function that brings the indicated axes (1-:-32) toward the target preset position value with a velocity profile toward 'S' time (with controlled jerk).

The consequence is smoother motion.

Syntax	**i32 mva_to_n_cj (i32 start, stru_mvitpcj stru [, i32 nDef=1] [, i32 holdFlag=0] [, real fr=1])**
start	Flag to enable data re-acknowledgement (only if axes stopped). The flag is reset by RTE when the data is taken
stru	Structure of type stru_mvitpcj 852 containing motion-related parameters
nDef	Size of the information array (number of axes involved). (optional, default 1)
holdFlag	Flag to command HOLD status to the mission. (optional, default 0)

© 2025 Robox SpA

Syntax	**i32 mva_to_n_cj (i32 start, stru_mvitpcj stru [, i32 nDef=1] [, i32 holdFlag=0] [, real fr=1])**
fr	Speed reduction value (0-1). (optional, default 1) See fr
Result	Variable containing information about the status of the function (see returns codes 61). <table><tr><th>Value</th><th>Description</th></tr><tr><td>-5 [M_ILLEGAL_ARGS]</td><td>Illegal parameter</td></tr><tr><td>-6 [M_IN_ALARM]</td><td>Alarm (ip axis already set)</td></tr><tr><td>2 [M_POSITIONING]</td><td>Initial positioning</td></tr><tr><td>6 [M_REACHED]</td><td>Target reached. This means that either the direction is 0 and the axis is stationary, or the direction is +1 or -1 and the axis has reached one of the end of strokes</td></tr><tr><td>14 [M_HOLDED]</td><td>Motion stopped due to HOLD flag. It is also returned in cases where the desired axis speed is imposed to 0 or the feedrate is set to zero</td></tr><tr><td>15 [M_ACCEL]</td><td>Acceleration in progress</td></tr><tr><td>16 [M_AT_SPEED]</td><td>Motion at speed</td></tr><tr><td>17 [M_DECEL]</td><td>Deceleration in progress</td></tr></table>
Validity	Rule
Note	The mission may involve multiple axes (max 32). They will be linearly interpolated in order to reach their targets all together

Diagnostics

- Generate alarm 9900 ill. Arg. A:2 T:x ST:xxx if invalid axis number or acceleration/speed/jerk < 1e-9
- Generate alarm 9900 ill. Arg. A:3 T:x ST:xxx if *acc* = 0

© 2025 Robox SpA

- Generate alarm 9903 Par. are changed T:x ST:xxx if running axis sequence changed
- Generate alarm 2800 Double Position Set T:x ST:xxx if already imposed axis ip

Examples

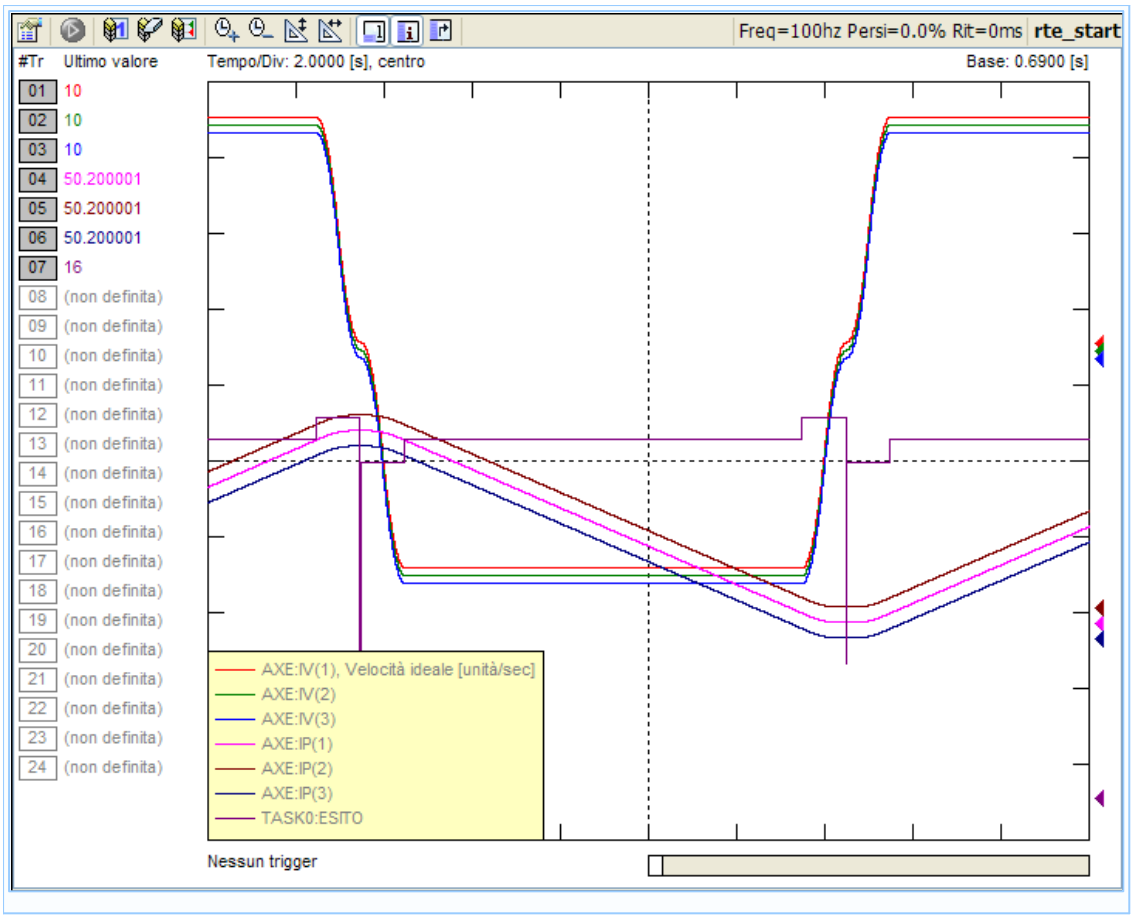
Example of use:

```
; example of three axes moving interpolated back and forth between
two points.
; definitions
stru_mvitpcj 852 m1[3]
int restartFlag

motion
if(first_time 321())
    rr(1000) = 100 ; target ax 1
    rr(1001) = 10  ; vel
    rr(1002) = 20  ; acc/dec
    rr(1003) = 40  ; jerk
    rr(2000) = 100 ; target ax 2
    rr(2001) = 10  ; vel
    rr(2002) = 20  ; acc/dec
    rr(2003) = 40  ; jerk
    rr(3000) = 100 ; target ax 3
    rr(3001) = 10  ; vel
    rr(3002) = 20  ; acc/dec
    rr(3003) = 40  ; jerkdec
    restartFlag = true
endif
; AXIS 1
;*********
; axis number
m1[0].nAx = 1
; desired position [units]
m1[0].target = rr(1000)
; desired speed [units/sec]
m1[0].v = rr(1001)
; acceleration/deceleration [units/sec²]
m1[0].acc = rr(1002)
; jerk [units/sec²]
m1[0].dec = rr(1003)
; dec hold value
m1[0].decH = max_acc(1)
; AXIS 2
;*********
; axis number
m1[1].nAx = 2
; desired position [units]
m1[1].target = rr(2000)
; desired speed [units/sec]
```

© 2025 Robox SpA

```
m1[1].v = rr(2001)
; acceleration/deceleration [units/sec²]
m1[1].acc = rr(2002)
; jerk [units/sec²]
m1[1].dec = rr(2003)
; dec hold value
m1[1].decH = max_acc(2)
; AXIS 3
;********
; axis number
m1[2].nAx = 3
; desired position [units]
m1[2].target = rr(3000)
; desired speed [units/sec]
m1[2].v = rr(3001)
; acceleration/deceleration [units/sec²]
m1[2].acc = rr(3002)
; jerk [units/sec²]
m1[2].dec = rr(3003)
; dec hold value
m1[2].decH = max_acc(3)
esito = mva_to_n_cj 475 (restartFlag, m1[0], 3, hold, fr)
if(rise 536 (esito = M_REACHED))
    if(avanti)
    inc 351 (rr(1000),100)
    inc(rr(2000),100)
    inc(rr(3000),100)
    avanti = false
    restartFlag = true
else
    dec 271 (rr(1000),100)
    dec(rr(2000),100)
    dec(rr(3000),100)
    avanti = true
    restartFlag = true
endif
end_motion
```

© 2025 Robox SpA



Function mva_to_n_v()

Function that moves the given axes (1-:-32) toward the target position value set with a velocity profile toward trapezoidal time.

Note: The difference with the mva_to_n 471() function is that it can be called at axis(s) already in motion and the final velocity on the target, also different from zero, can be specified.

Syntax	**i32 mva_to_n_v (i32 start, stru_mvgenitp stru [, i32 nDef=1] [, i32 holdFlag=0] [, real fr=1])**
start	Flag to enable the re-acknowledgement of data (even with moving axes). The flag is reset when the data is taken
stru	Structure of type stru_mvgenitp 849 containing motion-related parameters
nDef	Size of the information array (number of axes involved). (optional, default 1)

Syntax	**i32 mva_to_n_v (i32 start, stru_mvgenitp stru [, i32 nDef=1] [, i32 holdFlag=0] [, real fr=1])**
holdFlag	Flag to command HOLD status to the mission. (optional, default 0)
fr	Speed reduction value (0-1). (optional, default 1) See fr
Result	Variable containing information about the status of the function (see returns codes 61).

Value	Description
-5 [M_ILLEGAL_ARGS]	Illegal parameter*(acc=0* or illegal axis number)
-6 [M_IN_ALARM]	Alarm (axis ip already set)
6 [M_REACHED]	Target reached. This means that either the direction is 0 and the axis is stationary, or the direction is +1 or -1 and the axis has reached one of the end of strokes
14 [M_HOLDED]	Motion stopped due to HOLD flag. It is also returned in cases where the desired axis speed is imposed to 0 or the feedrate is set to zero
15 [M_ACCEL]	Acceleration in progress
16 [M_AT_SPEED]	Motion at speed
17 [M_DECEL]	Deceleration in progress

Validity	Rule
Note	The mission may involve multiple axes (max 32). They will be linearly interpolated in order to reach their targets all together. If the target is less than (pos_thr/10) away, RTE considers the space null. So in case the user wishes to make micro advances (<

© 2025 Robox SpA

Syntax	**i32 mva_to_n_v (i32 start, stru_mvgenitp stru [, i32 nDef=1] [, i32 holdFlag=0] [, real fr=1])**
	pos_thr/10) he will have to set the variable pos_thr appropriately

Examples

Example of use:

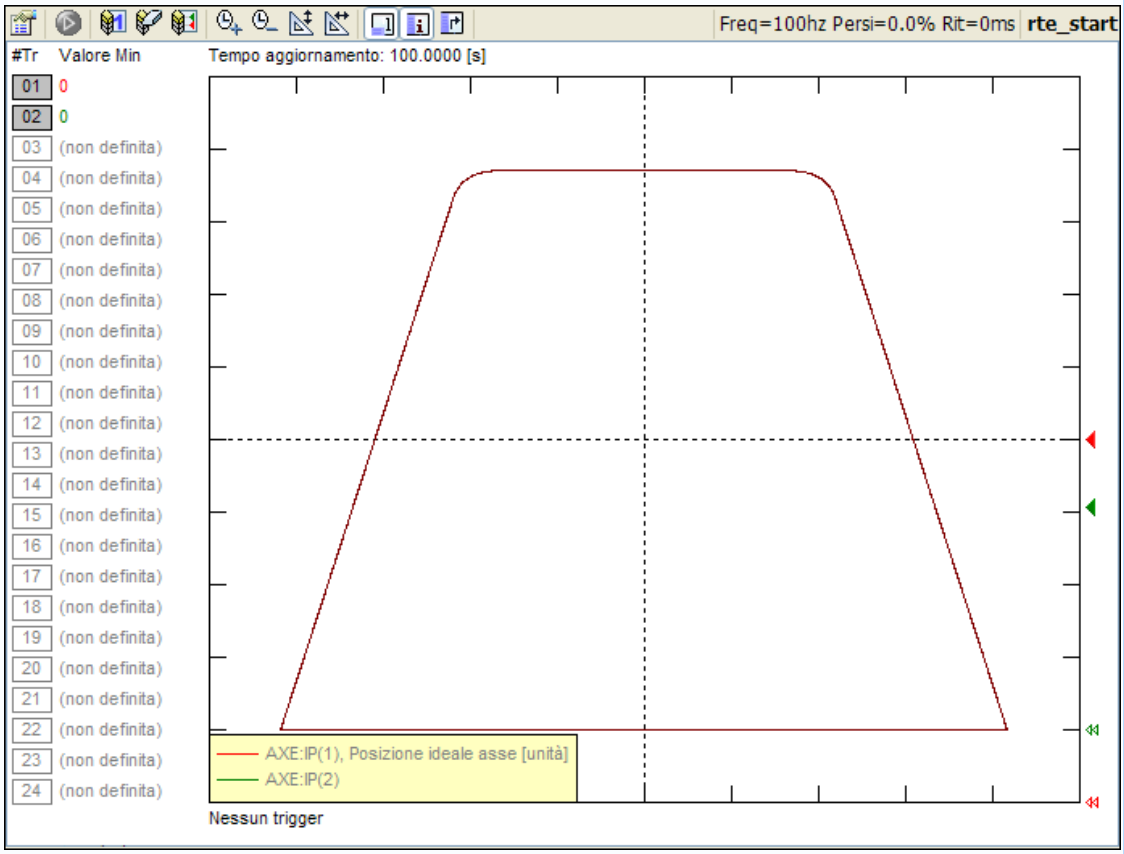
```
; execution in an x y plane of a trapezoid shape
; vertex A (0,0)
; vertex B (30,100)
; vertex C (70,100)
; vertex D (100,0)
; variable initializations
int fase, restartFlag, esito
stru_mvgenitp 849 m1[2]
; to be included in the rule
motion
if(first_time 321())
    fase = 1
endif

select(fase)
case 1  ; x moves toward point B
    rr(1000) = 30  ; target ax X
    rr(1001) = 10  ; vel regime
    rr(1002) = 10  ; vel fin
    rr(1003) = 10  ; acc ini
    rr(1004) = 0   ; acc fin

    ; y moves toward point B
    rr(2000) = 100 ; target ax Y
    rr(2001) = 10  ; vel regime
    rr(2002) = 0   ; vel fin
    rr(2003) = 10  ; acc ini
    rr(2004) = 0   ; acc fin
    restartFlag = true
    fase = 2
    break

case 2
    if(esito = M_REACHED)
        fase = 3
    endif
    break

case 3
    ; x moves toward point C
    rr(1000) = 70  ; target ax X
```

© 2025 Robox SpA

```
    rr(1001) = 10  ; vel regime
    rr(1002) = 10  ; vel fin
    rr(1003) = 10  ; acc ini
    rr(1004) = 0   ; acc fin
    ; y moves toward point C
    rr(2000) = 100 ; target ax Y
    rr(2001) = 0   ; vel regime
    rr(2002) = 0   ; vel fin
    rr(2003) = 10  ; acc ini
    rr(2004) = 0   ; acc fin
    restartFlag = true
    fase = 4
    break

case 4
    if(esito = M_REACHED)
        fase = 5
    endif
    break

case 5
    ; x moves toward point D
    rr(1000) = 100 ; target ax X
    rr(1001) = 10  ; vel regime
    rr(1002) = 0   ; vel fin
    rr(1003) = 10  ; acc ini
    rr(1004) = 0   ; acc fin
    ; y moves toward point D
    rr(2000) = 0   ; target ax Y
    rr(2001) = 10  ; vel regime
    rr(2002) = 0   ; vel fin
    rr(2003) = 10  ; acc ini
    rr(2004) = 0   ; acc fin
    restartFlag = true
    fase = 6
    break

case 6
    if(esito = M_REACHED)
        fase = 7
    endif
    break

case 7
    ; x moves toward point A
    rr(1000) = 0   ; target ax X
    rr(1001) = 10  ; vel regime
    rr(1002) = 0   ; vel fin
    rr(1003) = 10  ; acc ini
    rr(1004) = 0   ; acc fin
```

© 2025 Robox SpA

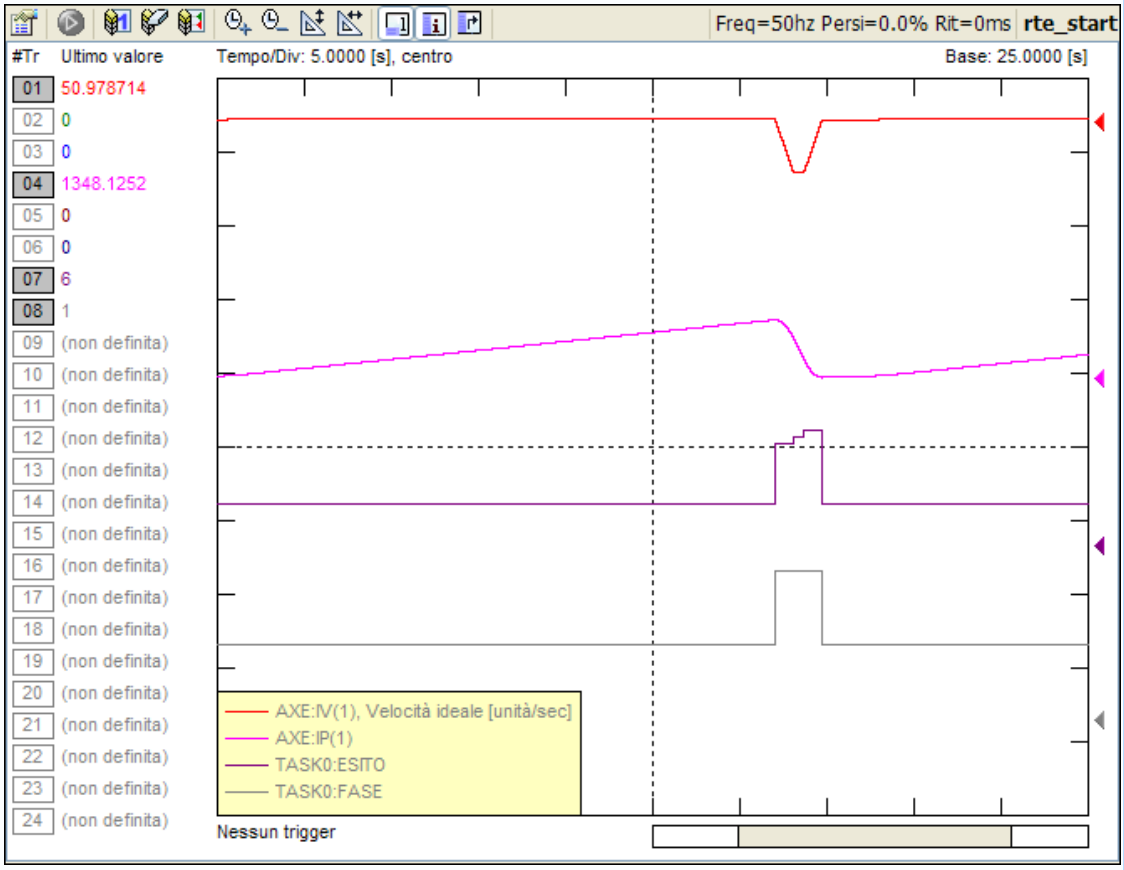
```
    ; y moves toward point A
    rr(2000) = 0    ; target ax Y
    rr(2001) = 0    ; vel regime
    rr(2002) = 0    ; vel fin
    rr(2003) = 10   ; acc ini
    rr(2004) = 0    ; acc fin
    restartFlag = true
    fase = 8
    break

case 8
    if(esito = M_REACHED)
        fase = 1
    endif
    break
end_select

; AXIS 1
;*********
; axis number
m1[0].nAx = 1
; desired position [units]
m1[0].target = rr(1000)
; initial speed [units/sec]
m1[0].v = rr(1001)
; final speed [units/sec]
m1[0].vf = rr(1002)
; initial acceleration [units/sec²]
m1[0].acci = rr(1003)
; final acceleration [units/sec²]
m1[0].accf = rr(1004)
; dec hold value
m1[0].accH = max_acc(1)
; AXIS 2
;*********
; axis number
m1[1].nAx = 2
; desired position [units]
m1[1].target = rr(2000)
; initial speed [units/sec]
m1[1].v = rr(2001)
; final speed [units/sec]
m1[1].vf = rr(2002)
; initial acceleration [units/sec²]
m1[1].acci = rr(2003)
; final acceleration [units/sec²]
m1[1].accf = rr(2004)
; dec hold value
m1[1].accH = max_acc(2)
esito = mva_to_n_v 479(restartFlag, m1[0] , 2, hold, fr)
end_motion
```



Esempio di utilizzo 2:

```
; on the rising edge input 1 the axis 1 that is traveling with its
law of motion returns
; rapidly to home quota with specified parameters
motion
if(first_time())
    fase = 1
endif
; if requested return to quota 0
if(rise(inp(1)))   ; return to base
    fase = 2
    ; axis number
    m1[0].nAx = 1
    ; desired position [units]
    m1[0].target = 0
    ; cruising speed [units/sec]
    m1[0].v = max_spe(1)
    ; final speed [units/sec]
    m1[0].vf = 0
    ; initial acceleration [units/sec²]
    m1[0].acci = max_acc(1)
    ; final acceleration [units/sec²]
```

```
    m1[0].accf = 0
    ; dec hold value
    m1[0].accH = max_acc(1)
    restartFlag = true ; assume data
endif
; cycle
if(fase = 1)   ; go with sinusoidal speed
    iv(1) = ramp(iv(1), 50 + sin(torad(mod(tfb*100, 360))),
10)
else    ; back to quota 0
    esito = mva_to_n_v(restartFlag, m1[0], 1, hold, fr)
    if(esito = M_REACHED)
        fase = 1
    endif
endif
end_motion
```



Function mva_zc()

Function that performs the axis zero cycle.

Syntax	**i32 mva_zc (i32 start, stru_mvzc stru [, i32 nDef=1] [, i32 holdFlag=0] [, real fr=1])**
start	Flag to enable data re-acknowledgement (only if axes stopped). The flag is reset by RTE when the data is taken
stru	Structure of type stru_mvzc 863 containing motion-related parameters
nDef	Size of the information array (number of axes involved). (optional, default 1)
holdFlag	Flag to command HOLD status to the mission. (optional, default 0)
fr	Speed reduction value (0-1). (optional, default 1) See fr
Result	Variable containing information about the status of the function (see returns codes 61).

Value	Description
-5 [M_ILLEGAL_ARGS]	Illegal parameter
-6 [M_IN_ALARM]	Alarm (ip axis already set)
6 [M_REACHED]	Zero cycle successfully terminated
7 [M_HOMING]	Zero cycle in execution

Validity	Rule
Note	If n axes are specified, the movements are simultaneous. It is RTE's responsibility to update the predefined variable c0_done if the zero cycle execution is terminated successfully. You can increase the diagnostics in the report via the predefined variable sys_flag.14

NOTE: The micro channel index should be specified in the project configurator -> axes -> transducer, it can be overridden via the predefined variable u0_channel.

NOTE: From RTE 34.23.5, the position preset value can be changed until it is actually used. Which means that the preset value set will be the last value written into the instruction parameters.

Diagnostics

- Generates alarm 9900 ill. A:1 T:x ST:xxx if invalid axis number
- Generate alarm 9900 ill. A:3 T:x ST:xxx if parameter *velPos* = 0
- Generates alarm 9900 ill. A:4 T:x ST:xxx if parameter *velZC* = 0
- Generate alarm 9900 ill. A:5 T:x ST:xxx if parameter *accel* = 0
- Generate alarm 9900 ill. A:6 T:x ST:xxx if parameter *jerk* = 0
- Generate alarm 9900 ill. A:8 T:x ST:xxx if programmed zero cycle is not applicable
- Generates alarm 9900 ill. A:9 T:x ST:xxx if flags are not congruent
- Generates alarm 9900 ill. A:10 T:x ST:xxx if the flags 'enable repositioning on preset quota'*(zcFlag*.9) and 'enable repositioning within software end of strokes'*(zcFlag*.10) are of exclusive type
- Generate alarm 9900 ill. A:11 T:x ST:if monodirectional movement is incompatible with request for positioning within end of strokes or on preset point
- Generates alarm 9900 ill. A:12 T:x ST:xxx if photo cannot be taken from device
- Generate alarm 9903 Par. are changed T:x ST:xxx if parameters are changed during movement
- Generate alarm 2800 #nAx Double Position Set T:x ST:xxx if already imposed the axis IP
- Generate alarm 2000#nAx Unable to perform homing

Types of homing

Axis type	Type of zero cycle
Linear axis	Micro zero cycle 66
Linear axis	Zero cycle at micro + index (incremental encoder transducer) 69
Linear axis	Zero cycle at end of strokes 72
Linear axis	Zero cycle at end of strokes + index (incremental encoder transducer) 74
Monodirectional rotary axis	Micro zero cycle with preliminary search for micro variation 83
Monodirectional rotary axis	Zero cycle at micro + index (incremental encoder transducer) with preliminary search for micro variation 76
Monodirectional rotary axis	Cycle of zero to micro without preliminary search for micro variation 85
Monodirectional rotary axis	Zero cycle at micro + index (incremental encoder transducer) without preliminary search for micro variation 80

© 2025 Robox SpA

Function normalize()

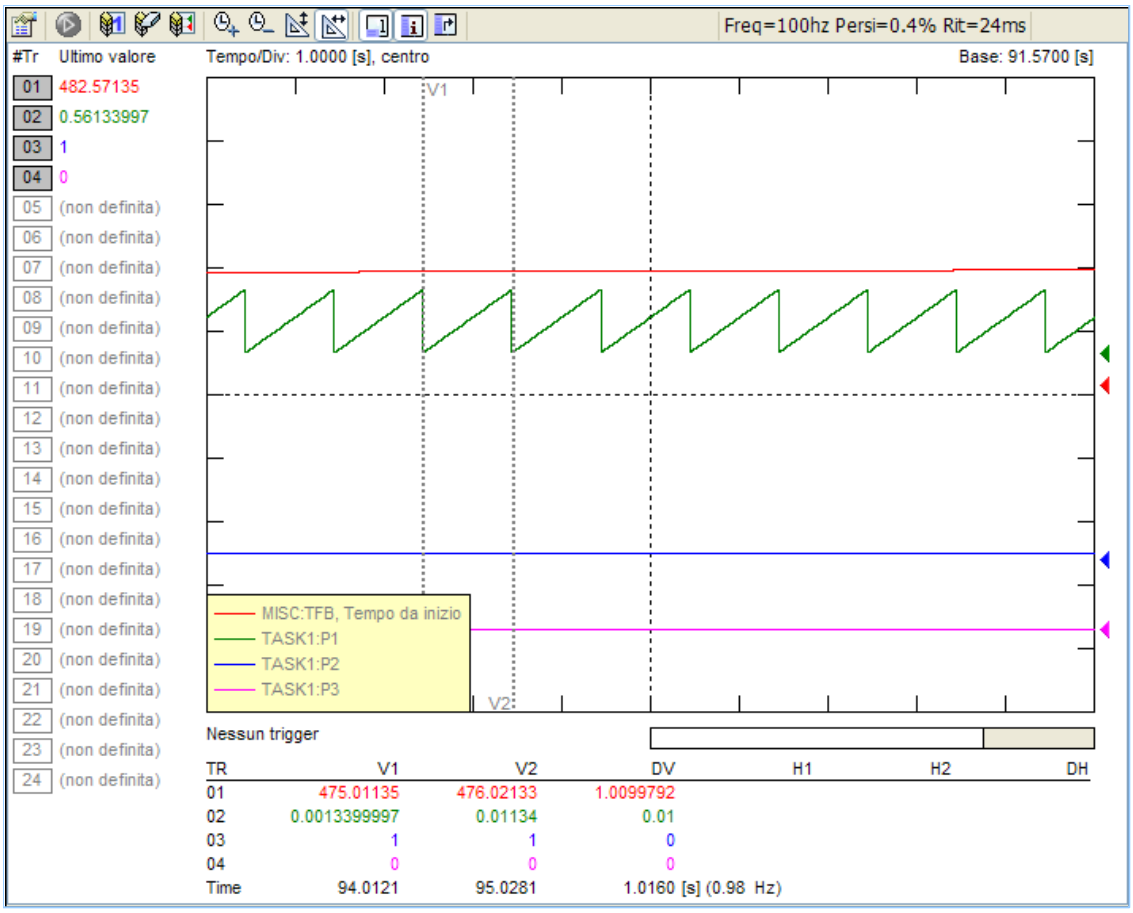
Function to calculate a normalized value between 0 and 1.

Syntax	**real normalize (real value, real offset, real cycle)**
value	Expression that provides the data to be normalized
offset	Expression giving the start position of the cycle
cycle	Expression providing the length of the reference cycle
Result	Returns a value between 0.0 (included) and 1.0 (excluded), calculated by the following formula: result = abs_mod 134 ((*value* - *offset*), *cycle*)
Validity	Rule, Task
Note	If the input parameter *value* exactly coincides with the start/end of a cycle, the normalize() function always returns 0.0 . The function therefore never returns the value 1.0

Examples

Example of use:

```
p1 = normalize 488 (tfb, p3, p2)   ; p3 = 0 e p2 = 1
```



Function onoff_filt()

Function to capture both the rising edge of a condition and the falling edge of a condition by setting filters in space or time.

Functioning

When the rising edge of the condition is detected, the value of the corresponding filter variable*(varFilt*) is stored.

As long as the difference between the current value of the filter variable and the stored value is less than the specified threshold*(thrTrue*) the function returns FALSE, while when the difference becomes greater than the specified threshold the function returns TRUE.

Only when the function is returning TRUE, at detection of the falling edge of the condition, the value of the corresponding filter variable*(varFilt*) is stored.

As long as the difference between the current value of the filter variable and the stored value is less than the specified threshold*(thrFalse*) the function continues to return TRUE, while when the difference becomes greater than the specified threshold the function returns FALSE.

© 2025 Robox SpA

Syntax	**bool onoff_filt (bool cond, real varFilt, real thrTrue [, real thrFalse=thrTrue])**
cond	Condition to be examined for front search
varFilt	Evolving quantity (typically time(tfb) or space ip(n))
thrTrue	Quantity of *varFilt* in which to stably see the true condition
thrFalse	Quantity of *varFilt* in which to stably see the false condition (optional, default *thrTrue*)
Result	Return:

Value	Condition
FALSE	At startup if *cond* = FALSE
	cond = FALSE and you have exceeded the threshold *thrFalse*
	cond = TRUE but the threshold *thrTrue* was not exceeded
TRUE	*cond* = TRUE and you have exceeded the threshold *thrTrue*
	cond = FALSE but the *thrFalse* threshold was not exceeded

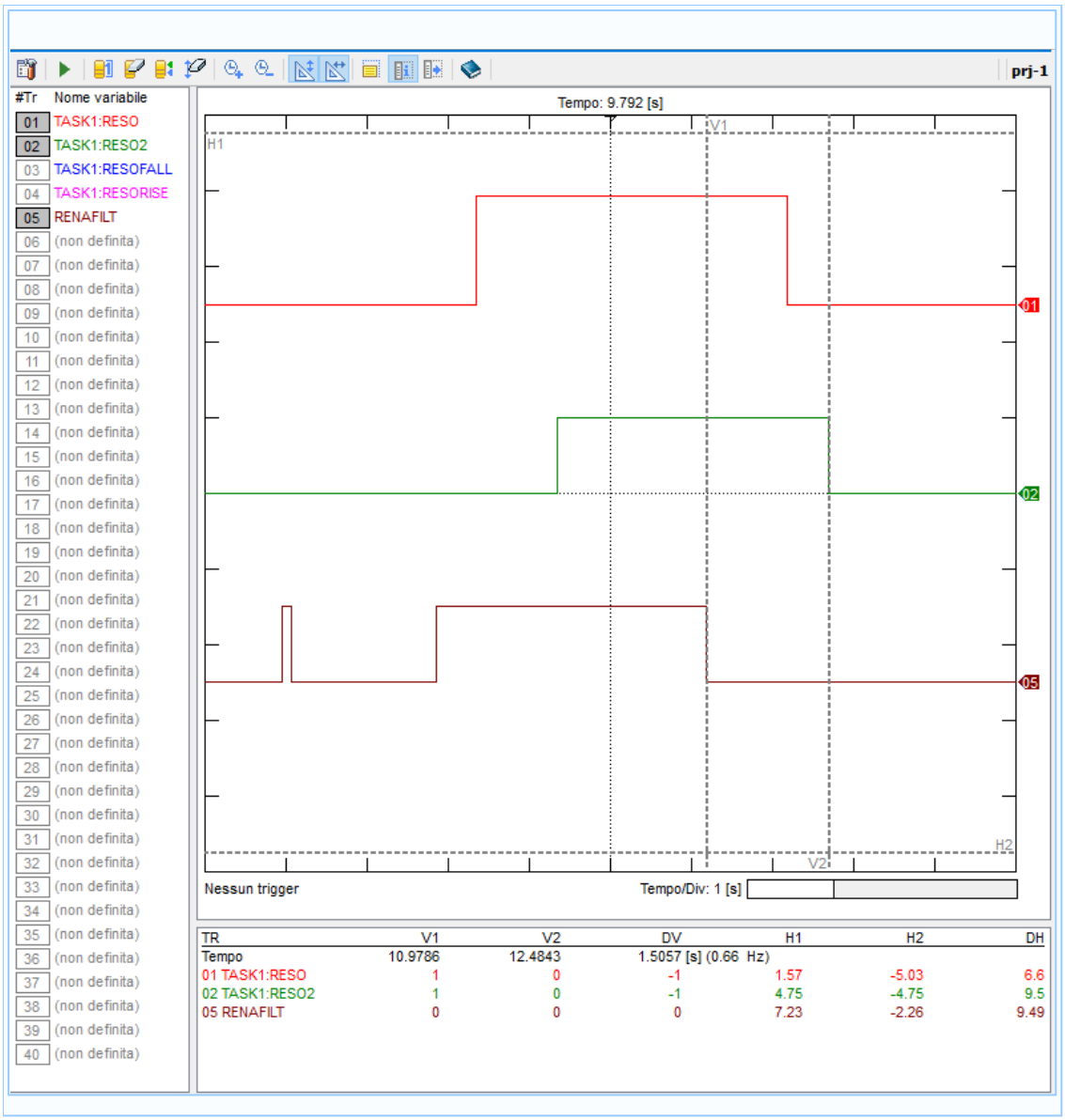
Validity	Rule, Task
Note	

Providing ip(n) as a filter variable, for example, will result a filter in space, while providing tfb as a filter variable will result a filter in time (seconds).

Examples

Example of use:

```
bool reso = false
bool reso2 = false

; after 0.5s since the condition is true, I set the output to TRUE,
; after 1s since the condition becomes false, I set the output to
FALSE
reso  = onoff_filt(rEnaFilt, tfb, 0.5, 1)
; I set the same threshold of 1.5s for both the rising and falling
edge
reso2 = onoff_filt(rEnaFilt, tfb, 1.5)
```



Function par_even()

Function that returns a TRUE value if the number of bits in the high state (1), in the indicated bytes of a variable, is even.

Syntax	**i32 par_even (i32 value [, i32 nByte=1])**
value	Value to calculate the parity of
nByte	Number of bytes. (optional, default 1)
Result	Returns:

Syntax	**i32 par_even (i32 value [, i32 nByte=1])**
	Value: > 0 — Number of bits at 1 (if even) Value: 0 — Otherwise
Validity	Rule, Task
Note	

Examples

Example of use:

```
if(par_even(r(100)))     ; 1 byte default
    ; Here we get there if the number of bits at 1 in the low byte of
R(100) is even
    r(101) = r(101)+1
end_if


if(par_even(r(100), 4))     ; all 4 bytes
    ; Here we get there if the number of bits to 1 in R(100) is even
    r(101) = r(101)+1
end_if
```

Function par_odd()

Function that returns a TRUE value if the number of bits in the high state (1), in the indicated bytes of a variable, is odd.

Syntax	**i32 par_odd (i32 value [, i32 nByte=1])**
value	Value to calculate the parity of
nByte	Number of bytes. (optional, default 1)
Result	Returns: Value: > 0 — Number of bits at 1 (if odd) Value: 0 — Otherwise
Validity	Rule, Task
Note	

Examples

Example of use:

```
if(par_odd(r(100)))     ; 1 byte default
    ; Here we get there if the number of bits at 1 in the low byte of
R(100) is odd
    r(101) = r(101)+1
end_if


if(par_odd(r(100), 4))     ; all 4 bytes
    ; Here we get there if the number of bits to 1 in R(100) is odd
    r(101) = r(101)+1
end_if
```

Function planar_comp_cp()

Function to apply reverse compensation in Planar Compensation Table(PCT) management.

Syntax	**i32 planar_comp_cp (stru_aax posIn, stru_aax posOut)**
posIn	Structure of type stru_aax 795 containing the CP positions of the axes
posOut	Structure of type stru_aax containing the compensated CP_C positions of the axes
Result	Always returns 0
Validity	Rule
System Requirements	RTE 34.16.0
Note	

Examples

Example of use:

```
; rule prologue 732
; -------------
struct stru_aax
    real axp[1+32]     ; array for positions
    ; index 0 is not used, so axp[1]....axp[32] is used
end_struct

; Sample code

; Global Storage definition
; -------------------------
stru_aax    in_cp, out_cp
stru_aax    in_ip, out_ip
```

```
...

; we use only 3 axes
in_ip.axp[1] = ip(1)
in_ip.axp[2] = ip(2)
in_ip.axp[3] = ip(3)
result = planar_comp_ip(in_ip, out_ip)

rr(120+1) = out_ip.axp[1]
rr(120+2) = out_ip.axp[2]
rr(120+3) = out_ip.axp[3]

in_cp.axp[1] = cp(1)
in_cp.axp[2] = cp(2)
in_cp.axp[3] = cp(3)

esito = planar_comp_cp(in_cp, out_cp)

rr(160+1) = out_cp.axp[1]
rr(160+2) = out_cp.axp[2]
rr(160+3) = out_cp.axp[3]
```

Function planar_comp_ip()

Function to apply direct compensation in Planar Compensation Table(PCT) handling.

Syntax	**i32 planar_comp_ip (stru_aax posIn, stru_aax posOut)**
posIn	Structure of type stru_aax containing the IP positions of the axes
posOut	Structure of type stru_aax containing the IP_C compensated positions of the axes
Result	Always returns 0
Validity	Rule
System Requirements	RTE 34.16.0
Note	

Examples

Example of use:

```
; rule_prologue
; -------------
struct stru_aax
    real axp[1+32]     ; array for positions
    ; index 0 is not used, so axp[1]....axp[32] is used
end_struct
```

```
; Sample code

; Global Storage definition
; -------------------------
stru_aax    in_cp, out_cp
stru_aax    in_ip, out_ip

...

; we use only 3 axes
in_ip.axp[1] = ip(1)
in_ip.axp[2] = ip(2)
in_ip.axp[3] = ip(3)
result = planar_comp_ip(in_ip, out_ip)

rr(120+1) = out_ip.axp[1]
rr(120+2) = out_ip.axp[2]
rr(120+3) = out_ip.axp[3]

in_cp.axp[1] = cp(1)
in_cp.axp[2] = cp(2)
in_cp.axp[3] = cp(3)

esito = planar_comp_cp(in_cp, out_cp)

rr(160+1) = out_cp.axp[1]
rr(160+2) = out_cp.axp[2]
rr(160+3) = out_cp.axp[3]
```

Function pos_capture_cmd()

This function enables the capture of an axis' quota photo via hardware (maximum precision).

Syntax	**i32 pos_capture_cmd (i32 nAx, i32 micro, i32 edge)**
nAx	Axis number (1-:-32)
micro	Transducer type:

Transducer selected	Value	Description
INCREMENTAL ENCODER	< 0	Disable capture
	0	Only channel of 0
	1	Micro of 0 associated

© 2025 Robox SpA

Syntax	**i32 pos_capture_cmd (i32 nAx, i32 micro, i32 edge)**

Transducer selected	Value	Description
		plus channel of 0
CANopen/EtherCAT Parker - SBC (HID type)	< 0	Disable Capture
	0	Immediate photo of resolver lap start or photo on channel
	1	input0 on drive
	2	input1 on drive
	3	input2 to drive
CANopen/EtherCAT Parker - SBC (SLVD/TPDM type)	< 0	Disable capture
	0	Immediate photo of resolver lap start or photo on channel
	1	input0 on drive. On TPDM, photos can be taken on multiple inputs if this performance is enabled
EtherCAT Robox IMD20	< 0	Disable Capture
	0	Immediate photo of start of resolver round or photo on channel
	1	input1 on drive
	2	input2 on drive
CANopen/EtherCAT Phase AXM	< 0	Disable capture

© 2025 Robox SpA

Syntax	**i32 pos_capture_cmd (i32 nAx, i32 micro, i32 edge)**

Transducer selected	Value	Description
(requires PLC program on board AXM)	0	Immediate photo of resolver lap start or photo on channel
	1	input0 on drive
	2	input1 on drive
	3	input2 to drive
	4	input3 to drive
CANopen/EtherCAT Phase Ultract TW	< 0	Disable capture. (not yet implemented)
	0	Immediate photo of resolver lap start or photo on channel
	1	input0 on drive
CANopen Sinamics RCB	-10 -11 -12 -13 -14 -15 -16	Hardware photo on specified input
	< 0 (different from above)	Disable capture
	0	Immediate photo of start of resolver round or photo on channel
	1 -:- 16	Software photo on specified input. (latency 1 ms)

edge

Bit flag:

Bit	Hex value	Bit status	Description
0	0x00000001	0	Falling edge. (if provided by hw)

© 2025 Robox SpA

Syntax	**i32 pos_capture_cmd (i32 nAx, i32 micro, i32 edge)**

Bit	Hex value	Bit status	Description
		1	Rising front. (if provided by hw) (used where applicable)
1	0x00000002	1	Continuous capture
2	0x00000004	1	Disable photo execution
3	0x00000008	1	Enables use of micro of 0 plus channel of 0 encoder. (if the hw has encoder channel read input)

Result

Returns:

Value	Meaning
> 0	Photo ID
0	Capture disabled
< 0	Error Code
-1	Photo hw not executable or error at photo startup or continuous mode not supported
-2	Photo already running
-6	Illegal axis number

Validity	Rule, Task
System Requirements	RTE 33.3.4
Note	

To be used in conjunction with the pos_capture_sts 499() function.

© 2025 Robox SpA

It can be used only with transducers that allow it ie:

- Incremental encoder
- CAN402 with Parker S.B.C. drive type (type HID 1) (requires cot hidrive_ipmode_3)
- CAN402 with Parker S.B.C. drive type (SLVD type 1) (requires cot slvdn_ipmode_3)
- EtherCAT with Parker S.B.C. drive type (needs Unique XML)
- EtherCAT/CANopen with Robox IMD20 drive type
- CAN402 with Sinamics drive type (RCB)
- EtherCAT/CANopen with Phase AXM drive type (requires PLC program loaded on AXM)
- EtherCAT/CANopen with drive type Phase TW (not yet implemented, does not require PLC program loaded on TW)

Function pos_capture_sts()

Support function to pos_capture_cmd 495() which is used to test whether the photo has been captured.

Syntax	**i32 pos_capture_sts (i32 nAx, i32 phId, real phDest [, i32 rawPh=NULL])**
nAx	Axis number (1-:-32)
phId	Value returned by pos_capture_cmd 495()
phDest	Variable in which the function writes the value of the captured quota
rawPh	Variable in which the function writes the raw value of the quota. (optional, default NULL)
Result	Returns:

Value	Meaning
1	Quota capture performed
0	System ready to capture quota
< 0	Error code
-1	Photo disabled by instruction or override
-2	Operation aborted due to command failure
-3	Override from loop of 0
-4	Error during photo of 0

© 2025 Robox SpA

Syntax	**i32 pos_capture_sts (i32 nAx, i32 phId, real phDest [, i32 rawPh=NULL])**
	Value / Meaning -5 / Internal error -6 / Illegal axis number -10 / ID no longer managed
Validity	Rule, Task
System Requirements	RTE 33.3.4, RC3E 33.1.9
Note	In case of continuous photo, only one report is given for each photo taken

Examples

Example of use:

```
; in the RULE_INIT or in the task before the LOOP.
; enables the capture of the axis quota on In0 on the Ascent one-shot
front (it self-disables after the first capture)
idFoto = pos_capture_cmd(asse, 1, 0x1)     ; idPhoto will take a
positive number representing the reservation of the photograph
; in the RULE
if(idFoto > 0)
    ; waits for the photo to be taken and returns its value in the
variable PhotoQuota
    EsitoFoto = pos_capture_sts(asse, IdFoto, FotoQuota)
    if(EsitoFoto)
        idFoto = pos_capture_cmd(asse, 1, 0x1)     ; rehabilitates
axis capture on In0 on the front of Ascent one-shot
 ; processing photograph of quota
    endif
endif
```

Function pow()

Function that calculates the power of a value.

Syntax	**real pow (real base, real exp)**
base	Base (must be > 0)
exp	Exponent
Result	Returns the *base* result elevated to *exp*
Validity	Rule, Task

© 2025 Robox SpA

Syntax	**real pow (real base, real exp)**
Note	If *base* is negative and *exp* is integer then performs power. If *base* is negative and *exp* is fractional returns 0 with error. If *exp* = 0 the function returns 1.0

Examples

Example of use:

```
result = pow(2,3)      ; result = 8
result = pow(2,0)     ; result = 1
result = pow(3,3)     ; result = 27
result = pow(10,-2)   ; result = 0.01
```

Function ps_channel_enable()

Function to command the enabling of group channels (PowerSet). The command can be valid for all axes or only for a subset.

NOTE: The reference to logical or physical axes depends on the one defined in configurator RTE.cfg.

Syntax	**i32 ps_channel_enable (power_set psName, i32 enaMsk [, i32 axMsk=0xFFFFFFFF])**
psName	PowerSet name (of type power_set)
enaMsk	Axis enable mask:

Bit	Hex value	Bit value	Description
0	0x00000001	0	Disable first axis of the group
		1	Enable first axis of the group
1	0x00000002	0	Disable second axis of the group
		1	Enable second axis of the group
2 -:- 30	0x000004 -:-	0	Disable n-th axis of the group

© 2025 Robox SpA

Syntax	**i32 ps_channel_enable (power_set psName, i32 enaMsk [, i32 axMsk=0xFFFFFFFF])**

Bit	Hex value	Bit value	Description
	0x40000000	1	Enable n-th axis of the group
31	0x80000000	0	Disable thirty-second axis of the group
		1	Enable thirty-second axis of the group.

NOTE: In case of physical axes, use the pam 896 property of the PowerSet.

axMsk

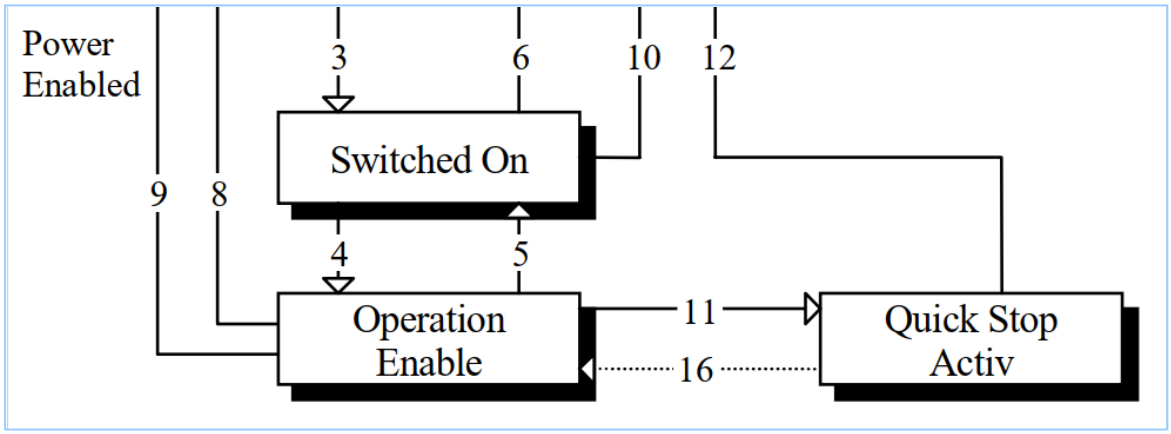
Involved axis mask: (optional, default 0xFFFFFF)

Bit	Hexadecimal value	Bit value	Description
0	0x00000001	0	Does not consider first axis of the group
		1	Manage first axis of the group
1	0x00000002	0	Does not consider second axis of the group
		1	Manage second axis of group
2 -:- 30	0x000004 -:- 0x40000000	0	Does not consider n-th axis of the group
		1	Manage n-th axis of the group

Syntax	**i32 ps_channel_enable (power_set psName, i32 enaMsk [, i32 axMsk=0xFFFFFFFF])**
	<table><tr><th>Bit</th><th>Hexadecimal value</th><th>Bit value</th><th>Description</th></tr><tr><td>31</td><td>0x80000000</td><td>0</td><td>Does not consider thirty-second axis of the group</td></tr><tr><td></td><td></td><td>1</td><td>Manage thirty-second axis of the group</td></tr></table>
Result	Returns: <table><tr><th>Value</th><th>Meaning</th></tr><tr><td>1</td><td>Successful operation</td></tr><tr><td>0</td><td>Otherwise</td></tr></table>
Validity	Rule, Task
Note	The function has no effect if the PowerSet of the axis in question has not been previously enabled with the ps_power_enable 514() instruction. The command variable used by RTE to perform the reset is specified by the user in the RTE.cfg file (project configurator)

In the case of axis managed in CAN402, the transition to operation enable (enabled) or switched on (disabled) is commanded via the control word.

With reference to the following image, if the axis is enabled, the requested transition will be number 4; if the axis is disabled, the requested transition will be number 5.



The behavior of a CAN402 drive, in case transition 5 is requested, is determined by the CANopen dictionary object 0x605C "disable operation option code."

Examples

Example of use:

```
power_set 896 robot1
result =  ps_channel_enable 501 (robot1, 0x4)  ; enables the third axis
of robot 1
result =  ps_channel_enable (robot1, 0xF)  ; enable the 4 axes of
robot 1
```

Function ps_channel_enabled()

Function to know the status of group channel enablements (PowerSet).

NOTE: The reference to logical or physical axes depends on the one defined in configurator RTE.cfg.

Syntax	**i32 ps_channel_enabled (power_set psName)**
psName	Name of the PowerSet (of type power_set 896).
Result	Mask axes that are in the enable state. In the case of axis managed in CAN402 it is the mask of drives in OPERATION ENABLE.

Bit	Hexadecimal value	Bit value	Description
0	0x00000001	0	First axis of the group disabled
		1	First axis of the group enabled
1	0x00000002	0	Second axis of the group disabled
		1	Second axis of the group enabled
2 -:- 30	0x000004 -:- 0x40000000	0	n-th axis of the group disabled

Syntax	**i32 ps_channel_enabled (power_set psName)**
	<table><tr><th>Bit</th><th>Hexadecimal value</th><th>Bit value</th><th>Description</th></tr><tr><td></td><td></td><td>1</td><td>n-th axis of the group enabled</td></tr><tr><td>31</td><td>0x80000000</td><td>0</td><td>Thirty-second axis of the group disabled</td></tr><tr><td></td><td></td><td>1</td><td>Thirty-second axis of the group enabled</td></tr></table>
Validity	Rule, Task
Note	

Examples

Example of use:

```
power_set 896 robot1
result = ps_channel_enabled 504 (robot1)
if(result r_and 0xF)
    sr(1) = "the first 4 axes are enabled!"
```

Function ps_channel_fault()

Function to know the fault status of the group channels.

NOTE: The reference to logical or physical axes depends on the one defined in configurator RTE.cfg.

Syntax	**i32 ps_channel_fault (power_set psName)**
psName	PowerSet name (of type power_set 896).
Result	Mask axes that are in a fault state. In the case of axis managed in CAN402 it is the mask of axes in fault state. <table><tr><th>Bit</th><th>Hexadecimal value</th><th>Bit value</th><th>Description</th></tr><tr><td>0</td><td>0x00000001</td><td>0</td><td>First axis of the group not</td></tr></table>

Syntax	**i32 ps_channel_fault (power_set psName)**

Bit	Hexadecimal value	Bit value	Description
			in fault state
		1	First axis of the group in fault state
1	0x00000002	0	Second axis of the group not in fault state
		1	Second axis of the group in fault state
2 -:- 30	0x000004 -:- 0x40000000	0	n-th axis of the group not in fault state
		1	n-th axis of the group in fault state
31	0x80000000	0	Thirty-second axis of the group not in fault state
		1	Thirty-second axis of the group in fault state

Validity	Rule, Task
Note	The command variable used by RTE to perform the reset is specified by the user in the RTE.cfg file (project configurator)

Examples

Example of use:

```
power_set 896 robot1
```

```
result = ps_channel_fault(robot1)
if(result r_and 0xF)
    sr(1) = "the first 4 axes are in fault!"
```

Function ps_channel_reset_fault()

Function to generate the group channel reset-fault command (PowerSet). The command can be valid for all axes or only for a subset.

NOTE: The reference to logical or physical axes depends on the one defined in configurator RTE.cfg.

Syntax	**i32 ps_channel_reset_fault (power_set psName, i32 rfMsk [, i32 axMsk=0xFFFFFF])**
psName	PowerSet name (of type power_set)
rfMsk	Mask of the reset fault commands:

Bit	Hexadecimal value	Bit Value	Description
0	0x00000001	0	First axis no reset fault command
		1	Reset fault first axis of the group
1	0x00000002	0	Second axis no reset fault command
		1	Reset fault second axis of the group
2 -:- 30	0x000004 -:- 0x40000000	0	n-th axis no reset fault command
		1	Reset fault n-th axis of the group
31	0x80000000	0	Thirty-second axis no reset fault command

Syntax	**i32 ps_channel_reset_fault (power_set psName, i32 rfMsk [, i32 axMsk=0xFFFFFF])**

Bit	Hexadecimal value	Bit Value	Description
		1	Reset fault thirty-second axis of the group

axMsk

Axis mask involved: (optional, default 0xFFFFFF)

Bit	Hexadecimal value	Bit value	Description
0	0x00000001	0	Do not consider first axis of the group
		1	Manage first axis of the group
1	0x00000002	0	Do not consider second axis of the group
		1	Manage second axis of the group
2 -:- 30	0x000004 -:- 0x40000000	0	Do not consider n-th axis of the group
		1	Manage n-th axis of the group
31	0x80000000	0	Do not consider thirty-second axis of the group
		1	Manage thirty-second axis of the group

© 2025 Robox SpA

Syntax	**i32 ps_channel_reset_fault (power_set psName, i32 rfMsk [, i32 axMsk=0xFFFFFF])**
Result	Returns: Value / Meaning 1 / Successful operation 0 / Otherwise
Validity	Rule, Task
Note	The command variable used by RTE to perform the reset is specified by the user in the RTE.cfg file (project configurator)

Examples

Example of use:

```
power_set 896 robot1
result = ps_channel_reset_fault 507(robot1, 0xF)   ; we give a reset
fault command to first 4 axes
```

Function ps_drive_reset()

Function to generate the drive-reset command to axes belonging to the group (PowerSet). The command can be valid for all axes or only for a subset.

NOTE: The reference to logical or physical axes depends on the one defined in configurator RTE.cfg.

Syntax	**i32 ps_drive_reset (power_set psName, i32 dfMsk [, i32 axMsk=0xFFFFFF])**
psName	Name of the powerSet (of type power_set 896)
dfMsk	Drive reset command mask:

Bit	Hexadecimal value	Bit value	Description
0	0x00000001	0	First axis no drive reset command
		1	Drive reset first axis of the group
1	0x00000002	0	Second axis no drive reset command

© 2025 Robox SpA

Syntax	**i32 ps_drive_reset (power_set psName, i32 dfMsk [, i32 axMsk=0xFFFFFF])**

Bit	Hexadecimal value	Bit value	Description
		1	Drive reset second axis of the group
2 -:- 30	0x000004 -:- 0x40000000	0	n-th axis no drive reset command
		1	Drive reset n-th axis of the group
31	0x80000000	0	Thirty-second axis no drive reset command
		1	Drive reset thirty-second axis of the group

axMsk

Axis mask involved: (optional, default 0xFFFFFF)

Bit	Hexadecimal value	Bit value	Description
0	0x00000001	0	Do not consider first axis of the group
		1	Manage first axis of the group
1	0x00000002	0	Do not consider second axis of the group
		1	Manage second axis of the group
2 -:- 30	0x000004 -:-	0	Do not consider n-

© 2025 Robox SpA

Syntax	**i32 ps_drive_reset (power_set psName, i32 dfMsk [, i32 axMsk=0xFFFFFF])**

Bit	Hexadecimal value	Bit value	Description
	0x40000000		th axis of the group
		1	Handle n-th axis of the group
31	0x80000000	0	Do not consider thirty-second axis of the group
		1	Manage thirty-second axis of the group

Result	Returns:

Value	Meaning
1	Successful operation
0	Otherwise

Validity	Rule, Task
Note	The command variable used by RTE to perform the reset is specified by the user in the RTE.cfg (project configurator) file

Examples

Example of use:

```
power_set 896 robot1
result = ps_drive_reset 509(robot1, 0x4)   ; we give a drive reset
command to axis 3
```

Function ps_exclude_lax()

Function to exclude a logical axis from the group (PowerSet).

Syntax	**i32 ps_exclude_lax (power_set psName, i32 nAx)**
psName	Name of the PowerSet (of type power_set 896)

Syntax	**i32 ps_exclude_lax (power_set psName, i32 nAx)**
nAx	Logical axis number
Result	Returns: Value — Meaning 0 — Successful operation -1 — PowerSet not existing -2 — Illegal axis number -3 — Axis not present in the specified PowerSet -4 — PowerSet energized, operation aborted -5 — Function not supported by RTE
Validity	Rule, Task
System Requirements	RTE 34.24.9
Note	Exclusion and re-inclusion operations are allowed if and only if the PowerSet is not energized. FB axes excluded from the PowerSet will be kept in the SHUTDOWN state and any diagnostics of them will not affect the behavior of the PowerSet itself

Examples

Example of use:

```
; in cfgdefs.i3
...
; {7:0} power set variables
...
power_set 896 sys
power_set ps_qua ; powerset definition
...

; in task or rule
int  result ; result of the function
int ax2manage ; axis number
...
result = ps_exclude_lax 511(ps_qua, ax2manage)
```

Function ps_exclude_pax()

Function to exclude a physical axis from the group (PowerSet).

Syntax	**i32 ps_exclude_pax (power_set psName, i32 nAx)**
psName	Name of the PowerSet (of type power_set 896)
nAx	Physical axis number
Result	Returns:

Value	Meaning
0	Successful operation
-1	PowerSet does not exist
-2	Illegal axis number
-3	Axis not present in the specified PowerSet
-4	PowerSet energized, operation aborted
-5	Function not supported by RTE

Validity	Rule, Task
System Requirements	RTE 34.24.9
Note	Exclusion and re-inclusion operations are allowed if and only if the PowerSet is not energized. FB axes excluded from the PowerSet will be maintained in the SHUTDOWN state and any diagnostics of them will not affect the behavior of the PowerSet itself

Examples

Example usage:

```
; in cfgdefs.i3
...
; {7:0} power set variables
...
power_set 896 sys
power_set ps_qua ; powerset definition
...

; in task or rule
int result ; result of the function
```

© 2025 Robox SpA

```
int ax2manage ; axis number
...
result = ps_exclude_pax(ps_qua, ax2manage)
```

Function ps_power_enable()

Function to enable group energization (PowerSet).

Syntax	**i32 ps_power_enable (power_set psName, i32 flag)**
psName	PowerSet name (of type power_set)
flag	Flag to enable the set: • 0, disable energization • 1, enable energization
Result	Returns: Value: 1 — Successful operation; 0 — Otherwise
Validity	Rule, Task
Note	To give the axis enable command, use the ps_channel_enable() instruction. The ps_channel_enabled() function has no effect if the PowerSet of the axis in question has not been previously enabled with the ps_power_enable() instruction. The command variable used by RTE to perform the reset is specified by the user in the RTE.cfg (project configurator) file. In the case of axes managed in CAN402, the transition to SWITCHED ON is commanded.

Examples

Example of use:

```
power_set robot1
if(robot1Present)
    result = ps_power_enable(robot1, 1)   ; Enables group
energization in accordance with what is defined in the RTE.CFG file
else
    result = ps_power_enable(robot1, 0)   ; disables group
energization
endif
```

© 2025 Robox SpA

Function ps_reinclude_lax()

Function to reinclude a logical axis in the group (PowerSet).

Syntax	**i32 ps_reinclude_lax (power_set psName, i32 nAx)**
psName	Name of the PowerSet (of type power_set 896)
nAx	Logical axis number
Result	Returns:

Value	Meaning
0	Successful operation
-1	PowerSet not existing
-2	Illegal axis number
-3	Axis not present in the specified PowerSet
-4	PowerSet energized, operation aborted
-5	Function not supported by RTE
-6	Axis already used in another PowerSet. Alarm 9909 is generated

Validity	Rule, Task
System Requirements	RTE 34.24.9
Note	Exclusion and re-inclusion operations are allowed if and only if the PowerSet is not energized. FB axes excluded from the PowerSet will be kept in the SHUTDOWN state and any diagnostics of them will not affect the behavior of the PowerSet itself

Examples

Example usage:

```
; in cfgdefs.i3
...
; {7:0} power set variables
...
```

© 2025 Robox SpA

```
power_set 896 sys
power_set ps_qua ; powerset definition
...

; in task or rule
int  result ; result of the function
int ax2manage ; axis number
...
result = ps_reinclude_lax 515(ps_qua, ax2manage)
...
```

Function ps_reinclude_pax()

Function to re-include a physical axis in the group (PowerSet).

Syntax	**i32 ps_reinclude_pax (power_set psName, i32 nAx)**
psName	Name of the PowerSet (of type power_set 896)
nAx	Physical axis number
Result	Returns:

Value	Meaning
0	Successful operation
-1	PowerSet not existing
-2	Illegal axis number
-3	Axis not present in the specified PowerSet
-4	PowerSet energized, operation aborted
-5	Function not supported by RTE
-6	Axis already used in another PowerSet. Alarm 9909 is generated

Validity	Rule, Task
System Requirements	RTE 34.24.9
Note	Exclusion and re-inclusion operations are allowed if and only if the PowerSet is not energized.

© 2025 Robox SpA

Syntax	**i32 ps_reinclude_pax (power_set psName, i32 nAx)**
	FB axes excluded from the PowerSet will be kept in the SHUTDOWN state and any diagnostics of them will not affect the behavior of the PowerSet itself

Examples

Example usage:

```
; in cfgdefs.i3
...
; {7:0} power set variables
...
power_set 896 sys
power_set ps_qua ; powerset definition
...

; in task or rule
int  result ; result of the function
int ax2manage ; axis number
...
result = ps_reinclude_pax 516(ps_qua, ax2manage)
...
```

Function ps_status()

Function to know the status of the group (PowerSet).

Syntax	**i32 ps_status (power_set psName)**
psName	PowerSet name (of type power_set 896)
Result	Returns:

Bit	Hexadecimal value	Description
0	0x00000001	At least one drive in fault
1	0x00000002	In power
2	0x00000004	At least one drive enabled
3	0x00000008	All drives are enabled
4	0x00000010	Status (possibly) delayed feedback

© 2025 Robox SpA

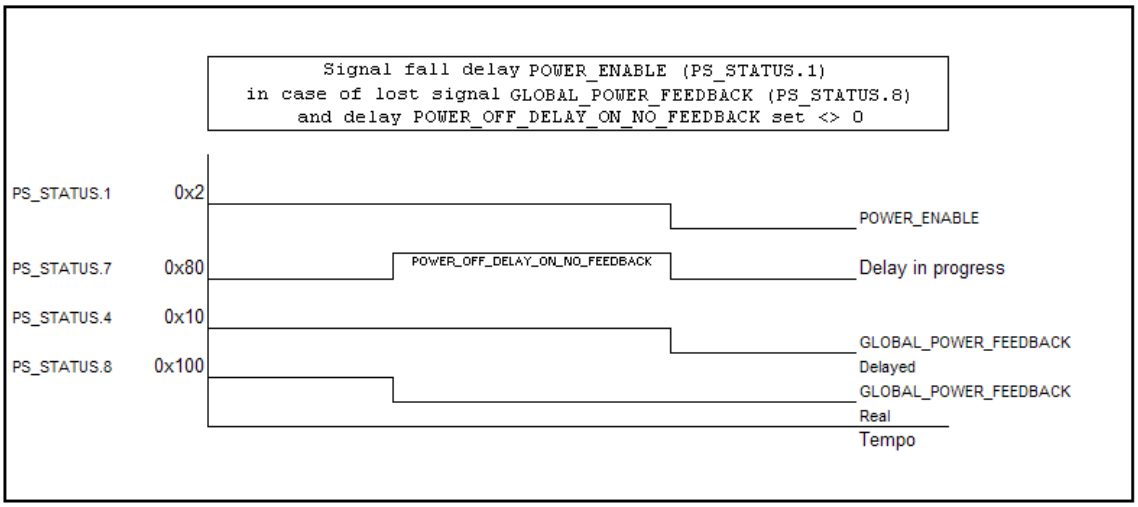
Syntax	i32 ps_status (power_set psName)

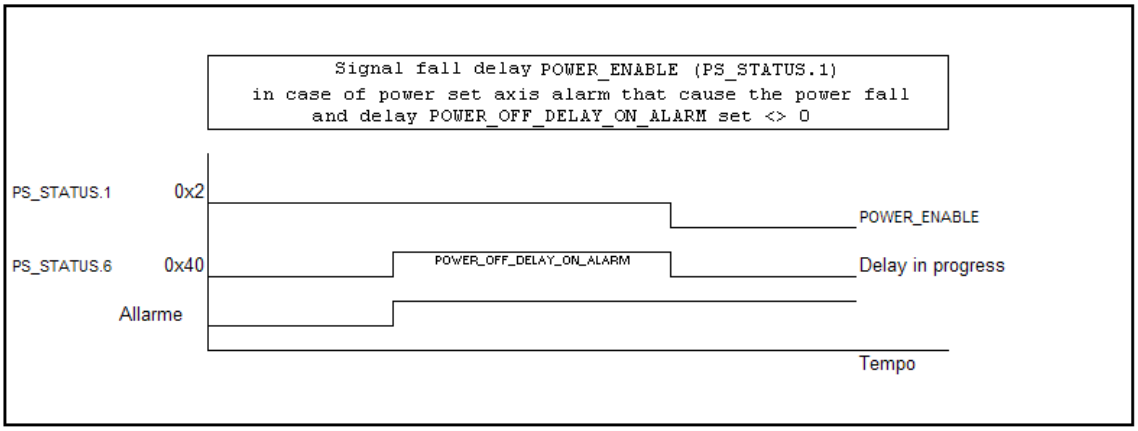
Bit	Hexadecimal value	Description
5	0x00000020	Reserved
6	0x00000040	Counting in progress on delay on PowerSet axis alarm causing power drop(power_off_delay_on_alarm)
7	0x00000080	Counting in progress in case of delay on no feedback of PowerSet(power_off_delay_on_no_feedback)
8	0x00000100	Actual state (not delayed) of feedback
9	0x00000200	Feedback is present for all drives in the PowerSet
10	0x00000400	If RTE version <= 34.27.01: feedback is present for all drives in the PowerSet that were commanded to be enabled(ps_channel_enable 501 ()). If RTE version >= 34.27.02: all enabled axes have the corresponding feedback and all enabled axes have the "drive follows command value" bit, bit 12 of fb_stsw

Syntax	**i32 ps_status (power_set psName)**

Bit	Hexadecimal value	Description
11	0x00000800	Excluded axes are present
12	0x00001000	PowerSet deenergized due to lack of global feedback. (from RTE 34.26.0)
13	0x00002000	All enabled axes have relative feedback
14	0x00004000	All enabled axes have the relative bit of "drive follows command value," bit 12 of fb_stsw
15 -:- 32	0x00008000 -:- 0x80000000	Reserved

Validity	Rule, Task
Note	RTE, to diagnose if the drive is in fault, tests the signal specified in the keyword ps_channel_fault 505()





Examples

Example of use:

```
power_set 896 robot1
int result
result =  ps_status 517 (robot1)
if(result r_and 777 0x8)
    sr(1) = "situazione ottima"
endif

if(result.6 or result.7)
    ; I manage braking and brake activation
endif
```

Function qwbe2lef()

Converts big endian qword (x64) to little endian real (x64).

Syntax	**real qwbe2lef (stru_large_int|large_uint stru)**
stru	Input structure of type stru_large_int 844 or type stru_large_uint 844
Result	Return value
Validity	Rule, Task
System Requirements.	RTE 34.24.10
Note	

Examples

Example of use:

```
stru_large_uint 844 input
real result

input.low  = 0xfb210940 ; Data memory: 40 09 21 fb hex.
input.high = 0x182d4454 ; Data memory: 54 44 2d 18 hex.
result = qwbe2lef 521(input)
; result = 3.141592653589793116 -> data memory: 18 2d 44 54 fb 21 09
40 hex.
```

Function qwbedwx2lef()

Converts dword exchange big endian qword (x64) to little endian real (x64).

Syntax	**real qwbedwx2lef (stru_large_int| large_uint stru)**
stru	Input structure of type stru_large_int 844 or type stru_large_uint 844
Result	Return value
Validity	Rule, Task
System Requirements	RTE 34.24.10
Note	

Examples

Example of use:

```
stru_large_uint 844 input
real result
```

© 2025 Robox SpA

```
input.low  = 0x182d4454 ; Data memory: 54 44 2d 18 hex.
input.high = 0xfb210940 ; Data memory: 40 09 21 fb hex.
result = qwbedwx2lef(input)
; result = 3.141592653589793116 -> data memory: 18 2d 44 54 fb 21 09
40 hex.
```

Function qwmbe2lef()

Converts middle big endian qword (x64) to little endian real (x64).

Syntax	**real qwmbe2lef (stru_large_int| stru_large_uint stru)**
stru	Input structure of type stru_large_int or type stru_large_uint
Result	Return value
Validity	Rule, Task
System Requirements	RTE 34.24.10
Note	

Examples

Example of use:

```
stru_large_uint input
real result

input.low  = 0x0940fb21 ; Data memory: 21 fb 40 09 hex.
input.high = 0x4454182d ; Data memory: 2d 18 54 44 hex.
result = qwmbe2lef(input)
; result = 3.141592653589793116 -> data memory: 18 2d 44 54 fb 21 09
40 hex.
```

Function qwmbedwx2lef()

Converts dword exchange middle big endian qword (x64) to little endian real (x64).

Syntax	**real qwmbedwx2lef (stru_large_int| stru_large_uint stru)**
stru	Input structure of type stru_large_int or type stru_large_uint
Result	Return value
Validity	Rule, Task

© 2025 Robox SpA

Syntax	**real qwmbedwx2lef (stru_large_int| stru_large_uint stru)**
System Requirements	RTE 34.24.10
Note	

Examples

Example of use:

```
stru_large_uint 844 input
real result

input.low  = 0x4454182d ; Data memory: 2d 18 54 44 hex.
input.high = 0x0940fb21 ; Data memory: 21 fb 40 09 hex.
result = qwmbedwx2lef 522(input)
; result = 3.141592653589793116 -> data memory: 18 2d 44 54 fb 21 09
40 hex.
```

Function r_rol()

Function that performs a left rotation of the specified number of bits.

Syntax	**i32 r_rol (i32 value, i32 nBit)**
value	Unit on which to perform the rotation
nBit	Number of bits
Result	Returns the result of the rotation
Validity	Rule, Task
Note	In the rotation the heaviest outgoing bit on the left is recirculated into the lightest incoming bit on the right

Examples

Example of use:

```
r(100) = r_rol 523(r(101), 3)
; Assuming a value of r(101) = 0x12481248, after execution of the
function the contents of r(100) will be : 0x92409240
```

Note: Since the heaviest bit does not exit to the left the behavior is analogous to r_shl 525().

Example usage:

```
r(100) = r_rol(r(101), 4)
; Assuming a value of r(101) = 0x12481248, after execution of the
instruction the contents of r(100) will be : 0x92409241
```

© 2025 Robox SpA

Note: The heaviest bit exits to the left and is recirculated to the right (different behavior from r_shl()).

Example usage:

```
r(1) = r_rol(r(1), 1)
```

# execution								
Initial value	0000	0000	0000	0000	0000	0000	0000	**0001**
1	0000	0000	0000	0000	0000	0000	0000	**0010**
2	0000	0000	0000	0000	0000	0000	0000	**0100**
...								
31	**1000**	0000	0000	0000	0000	0000	0000	0000
32	0000	0000	0000	0000	0000	0000	0000	**0001**

Function r_ror()

Function that performs a rotation to the right of the specified number of bits.

Syntax	**i32 r_ror (i32 value, i32 nBit)**
value	Unit on which to perform the rotation
nBit	Number of bits
result	Returns the result of the rotation
Validity	Rule, Task
Note	In rotation the lightest outgoing bit on the right is recirculated into the heaviest incoming bit on the left

Examples

Example usage:

```
r(100) = r_ror(r(101), 3)
; Assuming a value of r(101) = 0x12481248, after execution of the
function the contents of r(100) will be : 0x02490249
```

NOTE: Since the lightest bit does not exit to the right the behavior is similar to r_shr526().

Example usage:

```
r(100) = r_ror(r(101), 4)
; Assuming a value of r(101) = 0x12481248, after execution of the
instruction the contents of r(100) will be : 0x81248124
```

Note: The lightest bit exits to the right and is recirculated to the left (different behavior from r_shr()).

Example usage:

```
r(1) = r_ror(r(1), 1)
```

# execution								
Initial value	0000	0000	0000	0000	0000	0000	0000	**0001**
1	**1000**	0000	0000	0000	0000	0000	0000	0000
2	**0100**	0000	0000	0000	0000	0000	0000	0000
...								
31	0000	0000	0000	0000	0000	0000	0000	**0010**
32	0000	0000	0000	0000	0000	0000	0000	**0001**

Function r_shl()

Function that performs an arithmetic shift to the left of the specified number of bits.

Syntax	**i32 r_shl (i32 value, i32 nBit)**
value	Unit on which to perform the shift
nBit	Number of bits
result	Returns the result of the shift
Validity	Rule, Task
Note	In the left shift the heaviest bit exiting to the left is lost. A 0 enters on the right

Examples

Example usage:

```
r(100) = r_shl(r(101), 3)
; Assuming a value of r(101) = 0x12481248, after execution of the
function the contents of r(100) will be : 0x92409240
```

NOTE: Since the heaviest bit does not exit to the left the behavior is similar to r_rol() 523.

Example usage:

```
r(100) = r_rshl(r(101), 4)
; Assuming a value of r(101) = 0x12481248, after execution of the
instruction the contents of r(100) will be : 0x24812480
```

Note: The heaviest bit exits to the left and is lost. A 0 enters on the right (different behavior from r_rol()).

Example usage:

```
r(1) = r_shl(r(1), 1)
```

# execution								
Initial value	0000	0000	0000	0000	0000	0000	0000	**0001**
1	0000	0000	0000	0000	0000	0000	0000	**0010**
2	0000	0000	0000	0000	0000	0000	0000	**0100**
...								
31	**1000**	0000	0000	0000	0000	0000	0000	0000
32	0000	0000	0000	0000	0000	0000	0000	0000

Function r_shr()

Function that performs an arithmetic shift to the right of the specified number of bits.

Syntax	**i32 r_shr (i32 value, i32 nBit)**
value	Unit on which to perform the shift
nBit	Number of bits
result	Returns the result of the shift
Validity	Rule, Task
Note	In right shift the lightest bit exiting to the right is lost. In the heaviest bit a 0 enters if it was in 0 a 1 enters if it was in 1 (sign is kept)

Examples

© 2025 Robox SpA

Example usage:

```
r(100) = r_shr(r(101), 3)
; Assuming a value of r(101) = 0x12481248, after execution of the
function the contents of r(100) will be : 0x02490249
```

NOTE: Since the lightest bit does not exit to the right the behavior is similar to r_ror().

Example usage:

```
r(100) = r_rshr(r(101), 4)
; Assuming a value of r(101) = 0x12481248, after execution of the
instruction the contents of r(100) will be : 0x01248124
```

Example usage:

```
r(1) = r_shl(r(1), 1)
```

# execution								
Initial value	**1000**	0000	0000	0000	0000	0000	0000	**0001**
1	**1100**	0000	0000	0000	0100	0000	0000	0000
2	**1110**	0000	0000	0000	0010	0000	0000	0000
...								
31	**1111**	**1111**	**1111**	**1111**	**1111**	**1111**	**1111**	**1110**
32	**1111**	**1111**	**1111**	**1111**	**1111**	**1111**	**1111**	**1111**

Function r2m16()

Function used to manipulate a real value, in turn returning a real value, but modified so that it is in the field of valid values for a unit of type unsigned int 16-bit (equivalent to assigning an unsigned int 16 the incoming data and then copying this data into another real variable).

Syntax	**real r2m16 (real value)**
value	Is a generic expression of type real that provides the function with the data to manipulate
result	Returns the converted value
Validity	Rule, Task
Note	

Rules for converting data from real to unsigned 16 result in the following table:

Incoming value	Returned value
0	0
17	17
65535	65535
65536	0
65537	1
-1	65535
-17	65519
-65535	1
-65536	0
-65537	65535

Function r2m32()

Function used to manipulate a real value (double 8 bytes), still returning a real value, but modified so that it is returned to the field of valid values for a unit of type unsigned int 32 bits (equivalent to assigning an unsigned int 32 the incoming data and then copying this data to another real variable).

Syntax	**real r2m32 (real value)**
value	Is a generic expression of type real that provides the function with the data to manipulate
result	Returns the converted value
Validity	Rule, Task
Note	

Runs the module of *value* with the value 2^32 (4294967296):

- If *value* is positive, the function returns its module with the value 2^32
- If *value* is negative, the function executes its module with the value 2^32 of its absolute value. It then does the 2's complement of it

The rules for converting data from real to unsigned 32 result in the following table:

Value entered	Returned value
0	0
17	17
4294967295	4294967295
4294967296	0

© 2025 Robox SpA

Value entered	Returned value
4294967297	1
-1	4294967295
-17	4294967279
-4294967295	1
-4294967296	0
-4294967297	4294967295

Function r2m8()

Function used to manipulate a real value, still returning a real value, but modified so that it is in the field of valid values for a unit of type 8-bit unsigned int (equivalent to assigning an unsigned int 8 the incoming data and then copying this data into another real variable).

Syntax	**real r2m8 (real value)**
value	Is a generic expression of type real that gives the function the data to manipulate
result	Returns the converted value
Validity	Rule, Task
Note	

The rules for converting data from real to unsigned 8 result in the following table:

Incoming value	Returned value
0	0
17	17
255	255
256	0
257	1
258	2
-1	255
-17	239
-255	1
-256	0
-257	255
-258	254

© 2025 Robox SpA

Function ramp()

A function to linearly increment (build a ramp) a variable to a desired value.

For example, it can be useful to bring the ideal velocity of an axis to 0 with constant deceleration.

Syntax	**real ramp (real value, real target, real acc [, real dec=acc])**
value	Variable to be incremented (can be a register, a local variable, a kinematic unit, etc.)
target	Desired target value for the variable. The function will stop only when *value* equals *target*
acc	Acceleration value to reach *target*
dec	Deceleration value to reach *target*. (optional, default *acc*)
Result	Returns the updated value of the incoming variable
Validity	Rule
Note	Acceleration must mean value to move away from 0. Deceleration must mean value to move to 0

Examples

Example usage:

```
rr(3)= ramp(rr(3), 1000, 100)
; we ramp the variable rr(3) to the value 1000 with acc and dec equal
to 100.
; resort to the deltaVel/acc relationship to find out how much time
is taken.

rr(2) = ramp(rr(2), 1000, 100)
; if at the time of the call rr(2)=0 we will have an acceleration
time of (1000-0)/100 = 10 seconds. Same time to return to 0.

iv(AX_1) = ramp(iv(AX_1), rr(1), nvrr(1))
; we ramp axis 1 to the speed indicated in rr(1) with
acc/deceleration indicated in nvrr(1)
```

© 2025 Robox SpA

Function rand32()

Function that returns a random integer between 0 and MAX_INT32 (0x7fffff = +2147483647).

Syntax	**i32 rand32 ()**
Result	Returns the random number
Validity	Rule, Task
Note	

Examples

Example usage:

```
r(3000) = rand32()
```

Function random()

Function that returns a random integer between 0 and MAX_INT16 (0x7fff).

Syntax	**i16 random ()**
Result	Returns the random number
Validity	Rule, Task
Note	

Examples

Example usage:

```
midVal = midVal + error * random()/max_int16
```

Function random_value()

Function that returns a random real number between a given minimum and maximum value.

Syntax	**real random_value (real min, real max)**
min	Lower limit value
max	Upper limit value
Result	Returns the random number
Validity	Rule, Task
Note	

Examples

© 2025 Robox SpA

Example usage:

```
r(3000) = random_value(-100, 100)
```

Function range()

Function that tests if the value of the argument variable is within the desired limits.

Syntax	**i32 range (real value, real min, real max)**
value	Is the unit to be examined (can be a generic expression)
min	Is the lower bound value (can be a generic expression)
max	Is the upper limit value (can be a generic expression)
Result	Returns:

Value	Meaning
1	Value within the specified range
0	External value

Validity	Rule, Task
Note	

Examples

Example of use:

```
if(not range(rr(5), -12.5, +12.5))
    sr(1) = "the value assigned to rr(5) is not within the validity
range"
endif

if(not range(pluto, -12.5, +12.5))
    pluto = 12.5 * sign(pluto)
endif
; Assuming pluto = - 30 after 'Instruction execution pluto will equal
-12.5
```

Function range_adj()

Function that returns a scaling of the input parameter. Normalizes a variable that is in a given input range by bringing it into a given output range.

Syntax	**real range_adj (real inValue, real inLow, real inHigh, real outLow, real outHigh)**
inValue	Input value
inLow	Lower limit of the input variable
inHigh	Upper limit of the input variable
outLow	Lower limit of the normalized variable
outHigh	Upper limit of the normalized variable
Result	Returns the normalized value calculated as: ((*inValue* - *inLow*) /(*inHigh* - *inLow*)) *(*outHigh* - *outLow*) + *outLow*
Validity	Rule, Task
System Requirements	RC3E 33.9.0
Note	In the case where *inHigh* - *inLow* is equal to 0.0, the function returns *outLow*

Examples

Example of use:

```
lit nvrrMinSSI_bit nvrr(1000)
lit nvrrMaxSSI_bit nvrr(1001)
lit nvrrMaxSSI_mm  nvrr(1002)
lit nvrrMinSSI_mm  nvrr(1003)
lit rrRawSSI       rr(500)
lit rrNormField    rr(501)
...
nvrrMinSSI_bit = 3000
nvrrMaxSSI_bit = 6000
nvrrMinSSI_mm  = 500
nvrrMinSSI_mm  = 1000
...
rrRawSSI = 4500
rrNormField = range_adj(rrRawSSI, nvrrMinSSI_bit, &
                        nvrrMaxSSI_bit, nvrrMinSSI_mm, &
                        nvrrMinSSI_mm)
; ((4500-3000)/(6000-3000))*(1000-500)+500
; (1500/3000)*500+500
; 250+500
; rrNormField = 750
```

Function rbit()

Function to reset one or more bits in an integer-type data according to the occurrence of a condition.

Syntax	**i32 rbit (i32 data, i32 bitMsk, bool cond)**
data	Unit on which the reset is desired
bitMsk	Mask of the bits to be reset
cond	Condition upon occurrence of which bits will be reset
Result	Returns:

Value	Meaning
1	The condition is verified
0	Otherwise

Validity	Rule, Task
Note	

Examples

Example usage:

```
rbit534(r(1), 0x0000ff, r(100) = 34)
; Sets the low 8 bits to 0 when r(100) equals 34
```

Function release()

Function to release the forcing of I/O status.

Syntax	**i32 release (what)**
what	Object to be forced. Can be:

Type	Description
inp(nCh)	Where *nCh* is the input channel to be released. Can be any expression
inp_w(nW)	Where *nW* is the index of the input word to be released. Can be any expression
inp_w(nW).(iB)	Where *nW* is the index of the input

© 2025 Robox SpA

Syntax	**i32 release (what)**

Type	Description
	word and *iB* is the bit of the input word to be released. It can be any expression
out(nCh)	Where *nCh* is the output channel to be released. It can be any expression
out_w(nW)	Where *nW* is the index of the output word to be released. Can be any expression
out_w(nW).(iB)	Where *nW* is the index of the output word and *iB* is the bit of the output word to be released. It can be any expression

Result

Returns:

Value	Description
0	Operation performed successfully
-1	Version of RTE not compatible
-2	Invalid index
-3	Invalid bit

Validity	Rule, Task
Note	

Examples

Example of use:

```
$LIT iw_rem 401 ; remote word input index.
...
...
$LIT pbStart inp_w(iw_rem).(4) ; start pushbutton
...
...
esitoF = release 534 (pbStart, 0)
if(outcomeF < 0)
```

```
 ... ; diagnostics by error...
endif
```

Example usage:

```
$LIT iw_base_rem 401 ; remote word input index.
...
...
$LIT pbStop inp((iw_base_rem-1)*16+5) ; stop pushbuttom
...
...
esitoF = release(pbStop)
if(outcomeF < 0)
 ... ; diagnostics by error...
endif
```

Example usage:

```
; releases an output
; r(1000) contains the output channel
esitoF = release(out(r(1000)))
if(esitoF < 0)
 ... ; diagnostics by error...
endif
```

Function rise()

Function to capture the rising edge of a condition.

Syntax	**bool rise (cond)**
cond	Condition whose rising front is to be caught
Result	Returns TRUE only when the condition is verified and was FALSE at the previous execution
Validity	Rule, Task
Note	The first time it is executed, the function returns FALSE and stores the current state of the condition. In case rise 536() is used in a user R3 function, it should be avoided that the parameters of the function itself are involved in the condition expression. The result would not be as hoped because of the overlap in memory of the state of several variables

Each rise() function has its own memory of the condition state.

The evaluation of the front is done between the current state of the condition and the memory related to the individual function call (e.g., if we have at step 1000 `if(rise(inp(1 )))` and at step 1100 `if(rise(inp(1)))`, although they refer to the same condition, the evaluation is done separately, each according to the state stored by executing step 1000 and then step 1100).

© 2025 Robox SpA

Examples

Example of use:

```
if(rise 536(inp(449)))
    sr(1) = "fronte di salita"
endif

if(rise(rr(1)>12))
    sr(1) = "sopra la soglia"
endif
```

Function rise_filt()

Function to capture the rising edge of a condition by imposing a filter in space or time.

Functioning

Upon detection of the rising edge of the condition, the value of the corresponding filter variable is stored.

As long as the difference between the current value of the filter variable and the stored value is less than the specified threshold the function returns 0, while when the difference becomes greater than the specified threshold the function returns the stored value on the rising edge.

Syntax	**real rise_filt (bool cond, real varFilt, real thrFilt)**
cond	Condition to be examined for the front search
varFilt	Evolving unit (typically time (tfb) or space ip(n))
thrFilt	Quantity of *varFilt* in which the true condition is to be seen stably
Result	Returns:

Value	Condition
0	*cond* = FALSE
	cond = TRUE but the threshold *thrFilt* was not exceeded
<> 0 (the value of *varFilt* recorded on the rising edge of *cond* is returned)	*cond* = TRUE and you have exceeded *thrFilt* (reference variable value stored on transition)

Validity	Rule, Task

© 2025 Robox SpA

Syntax	**real rise_filt (bool cond, real varFilt, real thrFilt)**
Note	From the first time the function is executed until the condition holds high and the threshold is exceeded the value returned is 0. Past that threshold the value returned is the first supplied value of *varFilt*. In case rise_filt 537() is used in a user R3 function, it should be avoided that the parameters of the function itself are involved in the condition expression. The result would not be as hoped because of the overlap in memory of the state of several variables

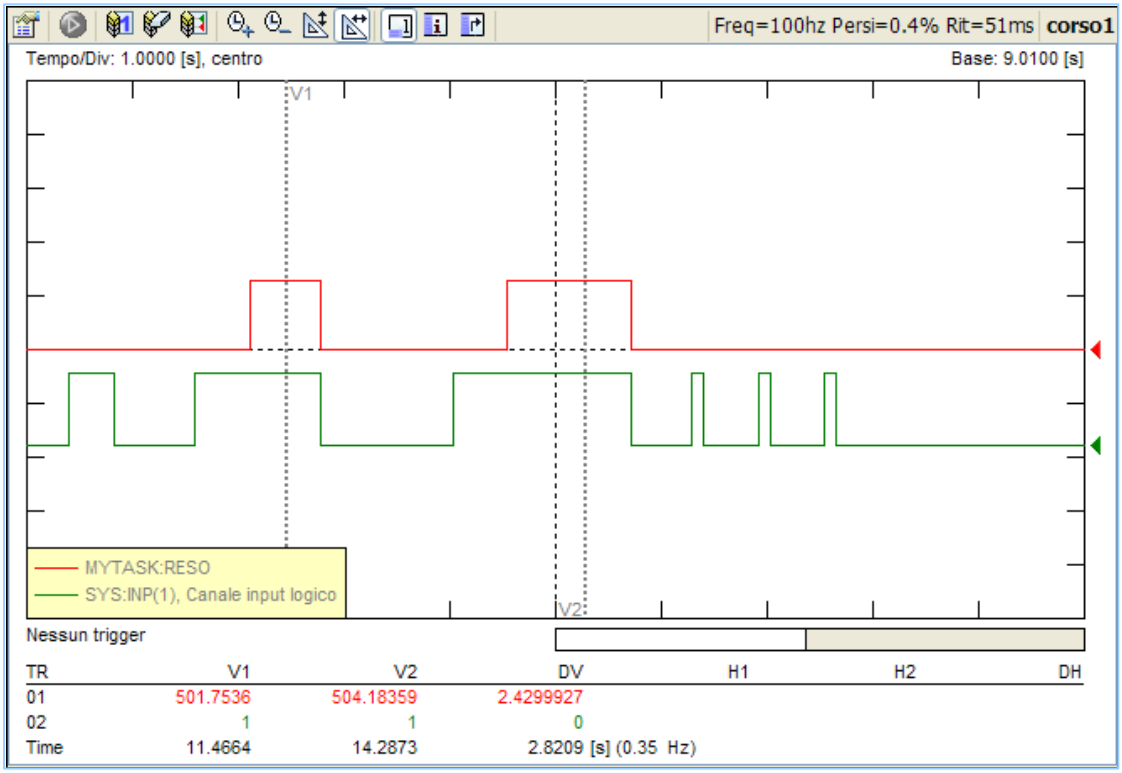
Providing ip(n) as a filter variable for example will result a filter in space, while providing tfb as a filter variable will result a filter in time (seconds).

Each function of rise_filt() has its own memory of the condition state. The front evaluation is done between the current state of the condition and the memory related to the individual function call. (e.g., if we have at step 1000 `if(rise_filt(inp(1), tfb, 0.5))` and at step 1100 `if(rise_filt(inp(1), tfb, 0.5))`, although they refer to the same condition, the evaluation is done separately, each according to the state stored by executing step 1000 and then step 1100).

Examples

Example of use:

```
reso = rise_filt(inp(1), tfb, 0.5)
```



```
; temporal filter on inp(1)
inp1f = not(fall_filt(rise_filt(inp(1), tfb, TimeFilter), tfb,
TimeFilter))

; calculation of the distance between the input rise and the current
position
; to manage an output between two thresholds
posUp = rise_filt(inp(1), ip(1), spaceFilter)
if(posUp)
    ; we are up with input. let's see the space traveled
    deltaPos = abs(ip(1) - posUp)
    ; open hatch to eliminate piece
    out(oApriBotola) = (deltaPos > MaxDimensionItem)
else
    out(oApriBotola) = false
endif
```

Function round()

Function that returns the integer closest to the number given.

Syntax	**i32 round (real value)**
value	Input value (can be negative)
Result	Integer number returned
Validity	Rule, Task
Note	

Examples

Example of use:

```
reso = round(12.3)     ; reso = 12
reso = round(12.499)  ; reso = 12
reso = round(12.5)    ; reso = 13
reso = round(12.9)    ; reso = 13
reso = round(-12.3)   ; reso = -12
reso = round(-12.499) ; reso = -12
reso = round(-12.5)   ; reso = -12
reso = round(-12.501) ; reso = -13
```

© 2025 Robox SpA

Function rtc_date()

Function to set the date of the real time clock/calender present on the connected axis controllers.

Syntax	**i32 rtc_date (i32 d, i32 m, i32 y)**
d	Variable containing the day (1-:-31)
m	Variable containing the month (1-:-12)
y	Variable containing the year (2007-:-2099)
Result	Returns:

Value	Meaning
0	Date successfully set
-1	Wrong day
-2	Wrong month
-3	Wrong year

Validity	Rule, Task
System Requirements.	RTE 33.8.1
Notes	The current clock/calender day, month, year, can be obtained via specific predefined variables such as day, month, year, ... The date can be displayed via date shell command

Examples

Example of use:

```
int reso
r(1)=20
r(2)=10
r(3)=2020
if(setDate)
    setdate = 0
    reso = rtc_date(r(1), r(2), r(3))   ; imposed date October 20,
2020
endif
```

Function rtc_time()

Function to set the time of the real time clock/calender present on the connected axis controllers.

Syntax	**i32 rtc_time (i32 h, i32 m, i32 s)**
h	Variable containing the time (0-:-23)

© 2025 Robox SpA

Syntax	**i32 rtc_time (i32 h, i32 m, i32 s)**
m	Variable containing the minutes (0-:-59)
s	Variable containing the seconds (0-:-59)
Result	Returns:

Value	Meaning
0	Time successfully set
-1	Wrong time
-2	Wrong minutes
-3	Wrong seconds

Validity	Rule, Task
System Requirements.	RTE 33.8.1
Notes	The current clock/calender hour, minute, second, ..., can be obtained via specific predefined variables such as hour, minute, second, ... The time can be displayed via shell command time

Examples

Example of use:

```
int reso
r(1)=20
r(2)=10
r(3)=59
if(setTime)
    setTime = 0
    reso = rtc_time(r(1), r(2), r(3))   ; imposed time 20:10:59
endif
```

Function safe_appl_ack()

Function to set the Application Acknowledge value of a safety island.

Syntax	**i32 safe_appl_ack (i32 nIsle, i32 value [, i32 mask=0xFFFFFF])**
nIsle	Number of the island (1-:-31)
value	Value to be set
mask	Mask of *value*. (optional, default 0xFFFFFF)
Result	Returns:

© 2025 Robox SpA

Syntax	**i32 safe_appl_ack (i32 nIsle, i32 value [, i32 mask=0xFFFFFF])**
	Value / Meaning 0 / Positive result -1 / Illegal parameters -2 / Illegal *nIsle* -4 / Non-existent island
Validity	Rule, Task
System Requirements	RTE 34.20.1
Note	

Examples

Example of use:

```
int esito
; imposed high the second Appl. Ack of the Island 3
esito = safe_appl_ack(3, 0x00000010)
if(esito = 0)
; Appl. setting. Ack successful
endif
```

Function safe_appl_diag()

Function that returns the status of the Application Diagnostic's bits of a safety island.

Syntax	**i32 safe_appl_diag (i32 nIsle [, i32 what=0])**
nIsle	Number of the island (1-:-31)
what	Parameter to select the bits group. (optional, default 0) Value / Meaning 0 / Receive all 32 bits 1 / Receive the first 16 bits 2 / Receive the last 16 bits
Result	Returns:

© 2025 Robox SpA

Syntax	**i32 safe_appl_diag (i32 nIsle [, i32 what=0])**

Condition	Value	Meaning
what = 0	> 0, Application Diagnostic (bit 0 -:- 31)	Positive result
	0	Negative result
what = 1	> 0, Application Diagnostic (bit 0 -:- 15)	Positive result
	-1	Illegal parameters
	-2	Illegal *nIsle*
	-4	Non-existent island
what = 2	> 0, Application Diagnostic (bit 16 -:- 31)	Positive result
	-1	Illegal parameters
	-2	Illegal *nIsle*
	-4	Non-existent island

Validity	Rule, Task
System Requirements	RTE 34.20.1
Note	

Examples

Example of use:

```
; Suppose that the state of the Application Diagnostics of island 1
is 0xaaaabbbb:
int esito
esito = safe_appl_diag 542(1)
; esito = 0xaaaabbbb
esito = safe_appl_diag(1, 1)
; esito = 0x0000aaaa
esito = safe_appl_diag(1, 2)
```

© 2025 Robox SpA

```
; esito = 0x0000bbbb
```

Function safe_diag()

Function that returns the diagnostic register of the desired satellite.

Syntax	**i32 safe_diag (i32 nIsle, i32 nSat)**
nIsle	Number of the island (1-:-31)
nSat	Satellite number (0-:-16)
Result	Returns:

Value	Module	Bit	Hex Value	Meaning
> 0	LPSDO8	16 -:- 31	0xFFFF0000	[Z] Zero
		15	0x8000	[COK] OK communication
		14	0x4000	[SA] Safety address written in bits 0-:-10
		13	0x2000	[E] Device in Error (override bit 14) error code written in bits 0-:-10. (see documentation "Phoenix Contact 106889 - Axioline F module with integrated safety logic and safe digital

Syntax	i32 safe_diag (i32 nIsle, i32 nSat)

Value	Module	Bit	Hex Value	Meaning
				outputs")
		12	0x1000	[PUR] Power Up request
		11	0x0800	[OAR] Operator Acknowledge request
		0 -:- 10	0x07FF	Safety address or error code
	SSDI8 - SSDO8	16 -:- 31		[Z] Zero
		15 - 14 - 13 (1 - 0 - 0)	0xE000 == 0x8XXX	[COK] OK communication
		15 - 14 - 13 (0 - 1 - 0)	0xE000 == 0x4XXX	[SA] Safety address written in bits 0-:-12
		15 - 14 - 13 (1 - 1 - 1)	0xE000 == 0xEXXXX X	[E] Device in Error, error code written in bits 0-:-12. (see module documentation)
		0 -:- 12	0x1FFF	Safety address or error code
-1				Illegal parameters

© 2025 Robox SpA

Syntax	**i32 safe_diag (i32 nIsle, i32 nSat)**
	Value / Module / Bit / Hex Value / Meaning -2 / Illegal *nIsle* -3 / *nSat* illegal -4 / Non-existent module
Validity	Rule, Task
System Requirements	RTE 34.20.1
Note	

Examples

Example of use:

```
; Suppose island 1 with an LPSDO8 as satellite 0 and an SSID8 as
satellite 1:
int diag_0, diag_1
diag_0 = safe_diag(1, 0)
diag_1 = safe_diag(1, 1)

; If everything works correctly we get:
; diag_0 = 0x00008000 (communication OK)
; diag_1 = 0x00008000 (communication OK)

; If satellite 1 receives a cross-circuit error on IN0_Ch2:
; diag_0 = 0x0000A000 (0x8000 of communication OK + 0x2000 of Device
Error +
; + 0x000 as the error was not received by him)
; diag_1 = 0x0000E128 (0xE000 of Device Error + 0x128 indicating the
type of error)
```

Function safe_diag_reset()

Function to send the Diagnostic Reset to the selected satellite.

Syntax	**i32 safe_diag_reset (i32 nIsle, i32 nSat)**
nIsle	Number of the island (1-:-31)
nSat	Satellite number (0-:-16) or -1 for all
Result	Returns:

© 2025 Robox SpA

Syntax	**i32 safe_diag_reset (i32 nIsle, i32 nSat)**
	Value / Meaning 0 / Positive result -1 / Illegal parameters -2 / Illegal *nIsle* -3 / *nSat* illegal -4 / Non-existent module
Validity	Rule, Task
System Requirements	RTE 34.20.1
Note	

Function safe_enab_out()

Function to set the value of the output enable register of the selected island.

Syntax	**i32 safe_enab_out (i32 nIsle, i32 nSat, i32 value [, i32 mask=0xFFFFFF])**
nIsle	Number of the island (1-:-31)
nSat	Satellite number (0-:-16)
value	Value to be setted
mask	Mask over *value* (optional, default 0xFFFFFF)
Result	Returns: Value / Meaning 0 / Positive result -1 / Illegal parameters -2 / Illegal *nIsle* -3 / *nSat* illegal -4 / Non-existent module
Validity	Rule, Task
System Requirements	RTE 34.20.1
Note	

Examples

© 2025 Robox SpA

Example of use:

```
; Suppose that in satellite island 1 satellite 3 there is an SSDO8
with the first 8 outputs high:
int esito
esito = safe_enab_out(1, 3, 0x1, 0xFF)
; only the first output is enabled and thus remains high, the others
have been disabled, that is, they are kept at 0 ("SAFE" status)
esito = safe_enab_out(1, 3, 0xFF, 0x0)
; by putting mask = 0 the outputs will not impose their
enabled/disabled status based on value,
; all enabled outputs will remain enabled and all disabled outputs
will remain disabled
```

Function safe_oper_ack()

Function to impose an Operation Acknowledge on the selected satellite.

Syntax	**i32 safe_oper_ack (i32 nIsle [, i32 nSat = 0])**
nIsle	Number of the island (1-:-31)
nSat	Satellite number (0-:-16). (optional, default 0)
Result	Returns:

Value	Meaning
0	Positive result
-1	Illegal parameters
-2	Illegal *nIsle*
-3	*nSat* illegal
-4	Non-existent module

Validity	Rule, Task
System Requirements	RTE 34.20.1
Note	

Examples

Example of use:

```
int esito
; set an Oper. Ack on satellite 3 of island 2
esito = safe_oper_ack(2, 3)
if(esito < 0)
    sr(1) = "Error in the setting of Oper. Ack"
```

© 2025 Robox SpA

```
endif
```

Function safe_prj_info()

Function to get information regarding the safety program running on the specified island.

Syntax	**i32 safe_prj_info (i32 nIsle, stru_safety_prjinfo stru)**
nIsle	Number of the isle (1-:-31)
stru	Data structure of type stru_safety_prjinfo 878 that will contain the project safety information
Result	Returns:

Value	Meaning
0	Positive result
-1	Illegal *nIsle*
-2	*stru* = NULL
-3	Non-existent island
-4	RTE version does not support this feature

Validity	Task
System Requirements	RTE 34.25.0
Note	

Examples

Example of use:

```
int esito_spi
stru_safety_prjinfo spi

esito_spi = safe_prj_info 549(1, spi)
if(esito_spi < 0)
    ; write here your diagnostic
else
    ; set value in registers for hmi
    sr(10) = spi.prjname
    r(10)  = spi.hb_crc
    r(11)  = spi.lb_crc
    r(12)  = spi.ab_crc
endif
```

© 2025 Robox SpA

Function safe_sts()

Function that returns the input/output safety status of the selected satellite.

Syntax	**i32 safe_sts (i32 nIsle, i32 nSat [, i32 what=0])**
nIsle	Number of the isle (1-:-31)
nSat	Number of the satellite (0-:-16)
what	Parameter to select the IO group. (optional, default 0) Value: Meaning 0: Any 1: Input only 2: Output only
Result	Returns: Value: Module: Meaning > 0: LPSDO8 - SSDO8: Output status SSDI8: Status of inputs -1: Illegal parameters -2: Illegal *nIsle* -3: *nSat* illegal -4: Non-existent module -5: Satellite not consistent with the requested type
Validity	Rule, Task
System Requirements	RTE 34.20.1
Note	

Examples

Example of use:

```
; Suppose we have an SSDO8 with the first 2 outputs at 1, on
satellite island 2 5
int esito
esito = safe_sts(2, 5)
; esito = 0x3
esito = safe_sts(2, 5, 1)
; esito = -5: I am asking for inputs to an SSDO8
```

Function sb0()

This function is used to read or write (instruction) byte 0 of the argument with sign extension.

Reading:

Syntax	**i8 sb0 (i8|i16|i32|u8|u16|u32|real arg)**
arg	Variable whose contents you want to know. From RTE 33.7.1 it can be any expression
Result	Returns the contents of the first byte of the argument *arg* with the sign
Validity	Rule, Task
Note	From RTE 33.9.0 access to a non-existing byte returns error

Writing (Instruction):

Syntax	**sb0 (i8|i16|i32|u8|u16|u32|real arg) = i8 value**
arg	Variable in which the *value* value will be imposed
value	Value to be imposed
Validity	Rule, Task
Note	From RTE 33.9.0 access to a non-existing byte returns error

Examples

Example of use:

```
I32 pluto
; if pluto equals to 0x8123
r(1) = sb0(pluto) ; r(1) = 0x0023
r(2) = sb1(pluto) ; r(2) = 0xff81
; comparison with b0 and b1
r(1) = b0(pluto)  ; r(1) = 0x0023
r(2) = b1(pluto)  ; r(2) = 0x0081
```

© 2025 Robox SpA

```
; if pluto 0x8234
r(1) = sb0(pluto) ; r(1) = 0x0034
r(2) = sb1(pluto) ; r(2) = 0xff82
; comparison with b0 and b1
r(1) = b0(pluto)  ; r(1) = 0x0034
r(2) = b1(pluto)  ; r(2) = 0x0082
; if pluto 0x1234
r(1) = sb0(pluto) ; r(1) = 0x0034
r(2) = sb1(pluto) ; r(2) = 0x0012
; comparison with b0 and b1
r(1) = b0(pluto)  ; r(1) = 0x0034
r(2) = b1(pluto)  ; r(2) = 0x0012

I16 appo,appo2,appo3
appo2 = 0x34
sb0(appo) = appo2  ; appo = 0x0034
sb1(appo3) = appo2 ; appo3 = 0x3400
; RTE 33.9.0:
I32 pippo = 123
sb4(pippo) = 9     ; error!
```

Function sb1()

This function is used to read or write (instruction) byte 1 of the argument with sign extension.

Reading:

Syntax	**i8 sb1 (i16|i32|u16|u32|real arg)**
arg	Variable whose contents you want to know. From RTE 33.7.1 it can be any expression
Result	Returns the contents of the second byte of the argument *arg* with the sign
Validity	Rule, Task
Note	From RTE 33.9.0 access to a non-existing byte returns error

Writing (Instruction):

Syntax	**sb1 (i16|i32|u16|u32|real arg) = i8 value**
arg	Variable in which the *value* value will be imposed
value	Value to be imposed
Validity	Rule, Task
Note	

© 2025 Robox SpA

Function sb2()

This function is used to read or write (instruction) byte 2 of the argument with sign extension.

Reading:

Syntax	**i8 sb2 (i32|u32|real arg)**
arg	Variable whose contents you want to know. From RTE 33.7.1 it can be any expression
Result	Returns the contents of the third byte of the argument *arg* with the sign
Validity	Rule, Task
Note	From RTE 33.9.0 access to a non-existing byte returns error

Writing (Instruction):

Syntax	**sb2 (i32|u32|real arg) = i8 value**
arg	Variable in which the *value* value will be imposed
value	Value to be imposed
Validity	Rule, Task
Note	

Function sb3()

This function is used to read or write (instruction) byte 3 of the argument with sign extension.

Reading:

Syntax	**i8 sb3 (i32|u32|real arg)**
arg	Variable whose contents you want to know. From RTE 33.7.1 it can be any expression
Result	Returns the contents of the fourth byte of the argument *arg* with the sign
Validity	Rule, Task
Note	From RTE 33.9.0 access to a non-existing byte returns error

Writing (Instruction):

© 2025 Robox SpA

Syntax	**sb3 (i32|u32|real arg) = i8 value**
arg	Variable in which the *value* value will be imposed
value	Value to be imposed
Validity	Rule, Task
Note	

Function sb4()

This function is used to read or write (instruction) byte 4 of the argument with sign extension.

Reading:

Syntax	**i8 sb4 (real arg)**
arg	Variable whose contents you want to know. From RTE 33.7.1 it can be any expression
Result	Returns the contents of the fifth byte of the argument *arg* with the sign
Validity	Rule, Task
Note	From RTE 33.9.0 access to a non-existing byte returns error

Writing (Instruction):

Syntax	**sb4 (real arg) = i8 value**
arg	Variable in which the *value* value will be imposed
value	Value to be imposed
Validity	Rule, Task
Note	

Function sb5()

This function is used to read or write (instruction) byte 5 of the argument with sign extension.

Reading:

Syntax	**i8 sb5 (real arg)**
arg	Variable whose contents you want to know. From RTE 33.7.1 it can be any expression

© 2025 Robox SpA

Syntax	**i8 sb5 (real arg)**
Result	Returns the contents of the sixth byte of the argument *arg* with the sign
Validity	Rule, Task
Note	From RTE 33.9.0 access to a non-existing byte returns error

Writing (Instruction):

Syntax	**sb5 (real arg) = i8 value**
arg	Variable in which the *value* value will be imposed
value	Value to be imposed
Validity	Rule, Task
Note	

Function sb6()

This function is used to read or write (instruction) byte 6 of the argument with sign extension.

Reading:

Syntax	**i8 sb6 (real arg)**
arg	Variable whose contents you want to know. From RTE 33.7.1 it can be any expression
Result	Returns the contents of the seventh byte of the argument *arg* with the sign
Validity	Rule, Task
Note	From RTE 33.9.0 access to a non-existing byte returns error

Writing (Instruction):

Syntax	**sb6 (real arg) = i8 value**
arg	Variable in which the *value* value will be imposed
value	Value to be imposed
Validity	Rule, Task
Note	

© 2025 Robox SpA

Function sb7()

This function is used to read or write (instruction) byte 7 of the argument with sign extension.

Reading:

Syntax	**i8 sb7 (real arg)**
arg	Variable whose contents you want to know. From RTE 33.7.1 it can be any expression
Result	Returns the contents of the eighth byte of the argument *arg* with the sign
Validity	Rule, Task
Note	From RTE 33.9.0 access to a non-existing byte returns error

Writing (Instruction):

Syntax	**sb7 (real arg) = i8 value**
arg	Variable in which the *value* value will be imposed
value	Value to be imposed
Validity	Rule, Task
Note	

Function sbit()

Function to set one or more bits in an integer-type data according to the occurrence of a condition.

Syntax	**i32 sbit (i32 data, i32 bitMsk, bool cond)**
data	Unit on which you want to set
bitMsk	Mask of the bits to be set
cond	Condition upon the occurrence of which the bits will be set
Result	Returns: Value — Meaning 1 — The condition is verified 0 — Otherwise
Validity	Rule, Task
Note	

Examples

Example of use:

```
sbit(r(1), 0x000000ff, r(100) = 34)
; Puts the low 8 bits at 1 when r(100) equals 34
; equals:
if(r(100) = 34))
    r(1) = r(1) r_or 0x000000ff
endif
```

Function sdw0()

This function is used to read or write (instruction) the dword 0 of the argument with sign extension.

Reading:

Syntax	**i32 sdw0 (i32|u32|real arg)**
arg	dword argument. From RTE 33.7.1 it can be any expression
Result	Returns the dword of the argument with the sign
Validity	Rule, Task
Note	From RTE 33.9.0 accessing a non-existing dword returns an error

Writing (Instruction):

Syntax	**sdw0 (i32|u32|real arg) = i32 value**
arg	dword argument
value	Value to be imposed in *arg*
Validity	Rule, Task
Note	From RTE 33.9.0 accessing a non-existing dword returns an error

Examples

Example of use:

```
REAL pippo1, pippo2, pippo3, pippo4
REAL pippo5, pippo6, pippo7, pippo8
INT prova[4]
prova[0] = 0x1122
prova[1] = 0x8001
prova[2] = 0x5566
prova[3] = 0x7788
```

© 2025 Robox SpA

```
pippo1 = sdw0 557 (prova[0]) ; pippo1 = -2147413726
pippo2 = sdw1 558 (prova[0]) ; pippo2 = 2005423462
pippo3 = sdw0(prova[2]) ; pippo3 = 2005423462
pippo4 = sdw1(prova[2]) ; pippo4 = 0
; Comparison with Instruction DW
pippo5 = dw0 275 (prova[0])  ; pippo1 = 2147553570
pippo6 = dw1 277 (prova[0])  ; pippo2 = 2005423462
pippo7 = dw0(prova[2])  ; pippo3 = 2005423462
pippo8 = dw1(prova[2])  ; pippo4 = 0

REAL pluto1, pluto2, pluto3, pluto4
REAL pluto5, pluto6, pluto7, pluto8
INT prova2[4]
prova2[0] = 0x1122
prova2[1] = 0x2233
prova2[2] = 0x5566
prova2[3] = 0x7788

pluto1 = sdw0(prova2[0]) ; pluto1 = 573772066
pluto2 = sdw1(prova2[0]) ; pluto2 = 2005423462
pluto3 = sdw0(prova2[2]) ; pluto3 = 2005423462
pluto4 = sdw1(prova2[2]) ; pluto4 = 0
; Comparison with Instruction DW (analogous)
pluto5 = dw0(prova2[0])  ; pluto1 = 573772066
pluto6 = dw1(prova2[0])  ; pluto2 = 2005423462
pluto7 = dw0(prova2[2])  ; pluto3 = 2005423462
pluto8 = dw1(prova2[2])  ; pluto4 = 0

; RTE 33.9.0:
I16 pippo = 123
sdw1(pippo) = 9  ; error!
```

Function sdw1()

This function is used to read or write (instruction) the dword 1 of the argument with sign extension.

Reading:

Syntax	**i32 sdw1 (real arg)**
arg	dword argument. From RTE 33.7.1 it can be any expression
Result	Returns the dword of the argument with the sign
Validity	Rule, Task

Syntax	**i32 sdw1 (real arg)**
Note	From RTE 33.9.0 accessing a non-existing dword returns an error

Writing (Instruction):

Syntax	**sdw1 (real arg) = i32 value**
arg	dword argument
value	Value to be imposed in *arg*
Validity	Rule, Task
Note	

Function ser_485_tx_sts()

Funzione che rende lo stato dell'ultima trasmissione fatta sul dispositivo specificato.

Syntax	**i32 ser_485_tx_sts (i32 nSer)**
nSer	Serial channel number
Result	Returns the status code. The meaning of the bits that make up the status code is:

Bit	Hexadecimal value	Meaning
0	0x00000001	485 transmission in progress
1	0x00000002	Transmission finished (sent last character)
2	0x00000004	Successful result - transmission finished and also received last character transmitted
3	0x00000008	Negative result - transmission aborted (cause bit 8)
4	0x00000010	Negative result - aborted transmission,

© 2025 Robox SpA

Syntax	**i32 ser_485_tx_sts (i32 nSer)**

Bit	Hexadecimal value	Meaning
		receive timeout
5	0x00000020	Free
6	0x00000040	Free
7	0x00000080	Transmission/reception errors
8	0x00000100	Negative result - serial not in 485 mode (12)
9	0x00000200	Negative result - serial not manageable in 485 mode

Validity	Rule, Task
Note	

Diagnostics

- Generates 9900 ill. Arg A:1 T:x ST:xxx if *nSer* illegal
- Generate 9901 ill. Funct. T:x ST:xxx if serial is not in 485 mode (see ser_mod 566)

Examples

Example of use:

```
function SendCommand()
    ; Serial 4
    ; Sending on the NUMSER serial lungh characters
    lungh = str_len 604(StrTx)
    ser_bin_out 563(NUMBER, StrTx, lungh)
    FotoTempo = tfb
    loopTx:
    if((ser_485_tx_sts 559(NUMSER) r_and 777 0x31C))
        ErroriTx = (ser_485_tx_sts(NUMSER) r_and 0x318)
        return
    endif
    if((tfb - FotoTempo) > 1.0)
        ; Timeout
        ErroriTx = 0x2000
```

```
        return
    endif
    goto loopTx
    return
end_fun
```

Function ser_bcc_h_init()

Function that performs the setting of data that is necessary to send a BCC message (for another Robox control) through serial channel.

Syntax	**i32 ser_bcc_h_init (bcc_msguser stru, i32 nSer, i32 nChDst, i32 nChSrc, i32 msgCode, i32 len)**
stru	Structure of type bcc_msguser 793 used for data to be sent. The structure of the data to be sent can be the same as the data received
nSer	Serial channel number (1, 2, 3, 4)
nChDst	Number of the user channel to be used as destination (190 -:- 199)
nChSrc	Number of the source user channel to be used for any replies (190 -:- 199)
msgCode	Code of the message (10000 -:- 19999)
len	Length of message data (max 255 bytes)
Result	Returns: Value / Meaning <> 0 / Operation performed correctly 0 / Otherwise
Validity	Rule, Task
Note	

Examples

Example usage:

```
; prepare serial 2 to transmit on channel 190 the structure block
with 5 Integer Registers bcc_msguser block
; variable of type structure bcc_msguser (it is a default structure)
reso = ser_bcc_h_init 561(block, 3, 190, 10000, 5*4) ; 5 * 4 bytes per
register
```

```
; block = structure
; 3 = use serial 3
; 190 = transmit on user channel 190
; 190 = if slave responds with bcc_h_setup() up use channel 190
; 10000 = code for this message 10000 (at user discretion in the
range reserved by the protocol)
; 5 * 4 = 20 bytes = 5 integer registers
```

Function ser_bin_in()

Function to read a specific number of characters from the specified device.

Syntax	**i32 ser_bin_in (i32 nSer, string dst, i32 nChar)**
nSer	Number of the serial channel
dst	Buffer in which to download the information
nChar	Number of bytes to read
Result	Returns:

Value	Meaning
<> 0	Operation performed correctly
0	Otherwise

Validity	Rule, Task
Note	

The function is non-blocking. If n characters are requested to be read and there are r characters in the receive buffer, the behavior is as follows:

- r < n, no characters are read and the function returns 0
- r >= n, exactly n characters are read and the function returns a TRUE value

Diagnostics

- Generates 9900 ill. Arg A:1 T:x ST:xxx if *nSer* illegal
- Generates 9901 ill. Funct. T:x ST:xxx if serial is not in 485 or single character mode (see ser_mod)

© 2025 Robox SpA

Function ser_bin_out()

Function to send a specific number of characters to the specified device.

Syntax	**i32 ser_bin_out (i32 nSer, string src, i32 nChar)**
nSer	Number of the serial channel
src	Buffer in which to download the information
nChar	Number of bytes to send
Result	Returns:

Value	Meaning
<> 0	Operation performed correctly
0	Otherwise

Validity	Rule, Task
Note	

Diagnostics

- Generates 9900 ill. Arg A:1 T:x ST:xxx if *nSer* illegal.
- Generates 9901 ill. Funct. T:x ST:xxx if serial is not in 485 or single character mode (see ser_mod 566)

Function ser_bin_ready()

Function that returns the number of characters present on the specified serial channel.

Syntax	**i32 ser_bin_ready (i32 nSer)**
nSer	Number of the serial channel
Result	Returns the number of characters
Validity	Rule, Task
Note	

Diagnostics

- Generates 9900 ill. Arg A:1 T:x ST:xxx if *nSer* illegal
- Generates 9901 ill. Funct. T:x ST:xxx if serial is not in 485 or single character mode (see ser_mod 566)

© 2025 Robox SpA

Function ser_buff_free()

Function that returns the number of free bytes in the transmission buffer of the specified serial.

Syntax	**i32 ser_buff_free (i32 nSer)**
nSer	Serial channel number
Result	Returns:

Value	Meaning
>= 0	Number of free bytes
< 0	Illegal *nSer*

Validity	Rule, Task
Note	

Function ser_buff_queue()

Function that returns the number of bytes occupied in the transmission buffer of the specified serial.

Syntax	**i32 ser_buff_queue (i32 nSer)**
nSer	Serial channel number
Result	Returns:

Value	Meaning
>= 0	Number of bytes occupied
< 0	Illegal *nSer*

Validity	Rule, Task
Note	

Function ser_flush()

Function that cleans all data in the TX and RX buffers of the specified serial.

Syntax	**i32 ser_flush (i32 nSer)**
nSer	Serial channel number
Result	Returns:

© 2025 Robox SpA

Syntax	**i32 ser_flush (i32 nSer)**
	Value / Meaning 0 / Operation performed successfully -2 / *nSer* illegal
Validity	Rule, Task
Note	

Function ser_in()

Function that moves a string from the receive buffer of the specified device to the destination string.

Syntax	**i32 ser_in (i32 nSer, string dst)**
nSer	Number of the serial channel
dst	Buffer into which to download the information
Result	Returns: Value / Meaning <> 0 / Operation performed correctly 0 / Otherwise
Validity	Rule, Task
Notes	The channel must necessarily be initialized as ascii (PSER_ASCII)

The string must be terminated by either the active terminator(terminator 750) or the default LF terminator (10 dec. - 0AH) or by reaching the maximum number of characters (83).

If in the specified buffer the string is not terminated then the function remains in idle state.

Diagnostics

- Generates 9900 ill. Arg A:1 T:x ST:xxx if *nSer* illegal
- Generates 9901 ill. Funct. T:x ST:xxx if serial is not in 485 or single character mode (see ser_mod 566)

© 2025 Robox SpA

Function ser_mod()

Questa funzione è usata per definire le modalità di lavoro dei dispositivi seriali.

Syntax	**i32 ser_mod (i32 nSer, i32 mod)**
nSer	Serial channel number (1, 2, 3, 4)
mod	Represents the chosen working mode for the serial channel:

Value	Mode
0	Robox protocol (DLE BCC) - with handshake (DTR-DSR) if supported by hardware
1	Pure serial communication - with handshake (DTR-DSR)
2	DLE-ASCII communication - without interlocks
12	RS485
17	Pure serial communication - without handshake
18	LFASCII communication - without handshake

Result	Returns:

Value	Meaning
<> 0	The port has been configured correctly
0	Otherwise

Validity	Rule, Task
Note	From RTE 34.22.01 if the Robox device serial does not have DTR and DSR, their handling is excluded (e.g., on SER2 of RP2)

Diagnostics

- 9900 ill. Arg A:1 T:x ST:xxx if illegal serial number or if the chosen mode is not compatible with the serial channel

Examples

© 2025 Robox SpA

Example of use:

```
int esito
; Programming serial channel 3 in 485 mode
esito = ser_mod(3, 12)
if(esito)
    sr(1) = "configuration performed correctly"
endif
```

Function ser_out()

Function that writes a string to the buffer of the specified device.

Syntax	**i32 ser_out (i32 nSer, string src)**
nSer	Number of the serial channel
src	String containing the data to be sent
Result	Returns: Value: Meaning <> 0: Operation performed correctly 0: Otherwise
Validity	Rule, Task
Note	The channel must necessarily be initialized as ascii (PSER_ASCII)

Diagnostics

- Generates 9900 ill. Arg A:1 T:x ST:xxx if *nSer* illegal
- Generate 9901 ill. Funct. T:x ST:xxx if serial is not in terminated string mode (see ser_mod)

Function ser_r_sts()

Function to read the status word of the specified serial channel.

Syntax	**i32 ser_r_sts (i32 nSer)**
nSer	Number of the serial channel
Result	Contains the status word of the serial channel as follows: Value: Meaning 1: DTR

© 2025 Robox SpA

Syntax	**i32 ser_r_sts (i32 nSer)**
	<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>2</td><td>RTS</td></tr><tr><td>4</td><td>DSR</td></tr><tr><td>8</td><td>CTS</td></tr><tr><td>16</td><td>All sent</td></tr><tr><td>32</td><td>CRC/framing error</td></tr><tr><td>64</td><td>RX overrun</td></tr><tr><td>128</td><td>Parity error</td></tr><tr><td>256</td><td>RX character available</td></tr><tr><td>512</td><td>TX buffer empty</td></tr></table>
Validity	Rule, Task
Note	You can only read the status of these signals if you have previously selected MODEM mode for the specified serial channel (see ser_mod 566)

Function ser_ready()

Function that checks if a terminated string is present in the buffer of the specified device.

Syntax	**i32 ser_ready (i32 nSer)**
nSer	Serial channel number
Result	Returns: <table><tr><th>Value</th><th>Meaning</th></tr><tr><td><> 0</td><td>A terminated string is present in the buffer of the specified device</td></tr><tr><td>0</td><td>Otherwise</td></tr></table>
Validity	Rule, Task
Note	The channel must necessarily be initialized as ascii (PSER_ASCII)

The string must be terminated by the active terminator (terminator 750) or the default LF terminator (10 dec. - 0AH) or by reaching the maximum number of characters (83).

Diagnostics

- Generates 9900 ill. Arg A:1 T:x ST:xxx if *nSer* illegal
- Generates 9901 ill. Funct. T:x ST:xxx if serial is not initialized as ascii

Function set_kinematics()

Function that simultaneously imposes the 3 kinematic units.

In case of limitation, the values are recalculated.

Syntax	**i32 set_kinematics (i32 nAx, real ipDes, real ivDes [, real iaDes=0])**
nAx	Axis number (1-:-32)
ipDes	Desired axis position [unit]
ivDes	Desired axis speed [unit/s]
iaDes	Desired axis acceleration [unit/s^2]. (optional, default 0)
Result	Returns:

Value	Meaning
-1	*nAx* illegal
0	Operation ok
1	Restricted due to IP
2	Restricted due to IV
3	Restricted due to IA

Validity	Rule
System Requirements	RTE 33.11.3
Note	For use when CAN is set in cubic interpolation mode (sending position and velocity targets)

Function set_virt_iw()

Function that sets the real value in a virtual input word.

The actual state of an input is defined as the physical state. The application will then read the physical state modified by any forcing.

Syntax	**i32 set_virt_iw (i32 nIW, i32 sts)**
nIW	Number of inp_w at which the actual state is set
sts	Value to be set
Result	Returns:

© 2025 Robox SpA

Syntax	**i32 set_virt_iw (i32 nIW, i32 sts)**
	Value: <> 0 — Meaning: Operation performed successfully Value: 0 — Meaning: The specified address does not belong to the defined virtual IO range
Validity	Rule, Task
Note	

Function shift_left()

Function that performs a logical shift to the left of one bit, in a specified window, of a u32 array, considered in fact as an array of bits.

Syntax	**i32 shift_left (u32 array, i32 idStart, i32 nBit, i32 stsBit)**
array	Reference array. **Note**: Even if of only one element it must be declared as a variable.
idStart	Index of the start bit of the window. The number must be between 0 and (arrayLength*32 - 1). **WARNING**: The user is responsible for checking the possible overflow.
nBit	Number of bits of window size
stsBit	Boolean value of the incoming bit
Result	Returns the value (0 or 1) of the outgoing bit (the highest) from the window in which the shift is performed
Validity	Rule, Task
Note	

Examples

Example of use:

```
u32 arr[2]
return = shift_left(arr, 32, 16, 1)
; at each execution on the second dword of the arr array the left
shift operation of one bit will be performed
```

		arr[1]								arr[0]							
Initial value		1010	1010	1010	1010	0000	0000	0000	0000	1010	1010	1010	1010	1010	1010	1010	1010
# execution	Return					Window (from index bit 32 by 16 bits)											
1	0	1010	1010	1010	1010	**0000**	**0000**	**0000**	**0001**	1010	1010	1010	1010	1010	1010	1010	1010
2	0	1010	1010	1010	1010	**0000**	**0000**	**0000**	**0011**	1010	1010	1010	1010	1010	1010	1010	1010
...	0	1010	1010	1010	1010	**...**	**...**	**...**	**...**	1010	1010	1010	1010	1010	1010	1010	1010
15	0	1010	1010	1010	1010	**0111**	**1111**	**1111**	**1111**	1010	1010	1010	1010	1010	1010	1010	1010
16	0	1010	1010	1010	1010	**1111**	**1111**	**1111**	**1111**	1010	1010	1010	1010	1010	1010	1010	1010
17	1	1010	1010	1010	1010	**1111**	**1111**	**1111**	**1111**	1010	1010	1010	1010	1010	1010	1010	1010

Function shift_right()

Function that performs a logical shift to the right of one bit, in a specified window, of a u32 array, considered in fact as an array of bits.

Syntax	**i32 shift_right (u32 array, i32 idStart, i32 nBit, i32 stsBit)**
array	Reference array. **Note**: Even if of only one element it must be declared as a variable.
idStart	Index of the start bit of the window. The number must be between 0 and (arrayLength*32 - 1). **WARNING**: The user is responsible for checking the possible overflow.
nBit	Number of bits of window size
stsBit	Boolean value of the incoming bit

© 2025 Robox SpA

Syntax	**i32 shift_right (u32 array, i32 idStart, i32 nBit, i32 stsBit)**
Result	Returns the value (0 or 1) of the outgoing bit (the lowest) from the window in which the shift is performed
Validity	Rule, Task
Note	

Examples

Example of use:

```
u32 arr[2]
return = shift_right(arr, 16, 16, 1)
; at each execution on the second word of the arr array will be
performed the one-bit right shift operation
```

		arr[1]								arr[0]							
Initial value:		1010	1010	1010	1010	0000	0000	0000	0000	1010	1010	1010	1010	1010	1010	1010	1010
# execution	Return									Window (from index bit 16 by 16 bits)							
1	0	1010	1010	1010	1010	1010	1010	1010	1010	**1000**	**0000**	**0000**	**0000**	1010	1010	1010	1010
2	0	1010	1010	1010	1010	1010	1010	1010	1010	**1100**	**0000**	**0000**	**0000**	1010	1010	1010	1010
...	0	1010	1010	1010	1010	1010	1010	1010	1010	**...**	**...**	**...**	**...**	1010	1010	1010	1010
15	0	1010	1010	1010	1010	1010	1010	1010	1010	**1111**	**1111**	**1111**	**1110**	1010	1010	1010	1010
16	0	1010	1010	1010	1010	1010	1010	1010	1010	**1111**	**1111**	**1111**	**1111**	1010	1010	1010	1010
17	1	1010	1010	1010	1010	1010	1010	1010	1010	**1111**	**1111**	**1111**	**1111**	1010	1010	1010	1010

© 2025 Robox SpA

Function shl()

Function that performs a logical shift to the left of the specified number of bits.

Syntax	**i32 shl (i32 value, i32 nBit)**
value	Variable to perform the shift on
nBit	Number of bits to be shifted
result	Returns the result of the shift
Validity	Rule, Task
Note	In the left shift the heavier outgoing bit is lost, in the lighter bit 0 always enters. It behaves similarly to r_shl 525()

Examples

Example of use:

```
r(1) = shl 573(r(1), 1)
```

# execution								
Initial value	0000	0000	0000	0000	0000	0000	0000	000**1**
1	0000	0000	0000	0000	0000	0000	0000	00**1**0
2	0000	0000	0000	0000	0000	0000	0000	0**1**00
...								
31	**1**000	0000	0000	0000	0000	0000	0000	0000
32	0000	0000	0000	0000	0000	0000	0000	0000

Function shr()

Function that performs a logical shift to the right of the specified number of bits.

Syntax	**i32 shr (i32 value, i32 nBit)**
value	Variable on which to perform the shift
nBit	Number of bits to be shifted
result	Returns the result of the shift
Validity	Rule, Task

© 2025 Robox SpA

Syntax	**i32 shr (i32 value, i32 nBit)**
Note	In right shift the lightest outgoing bit is lost, the heaviest bit always enters 0 (sign is lost). To keep the sign use r_shr 526()

Examples

Example of use:

```
r(1) = shr 573(r(1), 1)
```

# execution								
Initial value	**1**000	0000	0000	0000	**1**000	0000	0000	000**1**
1	0**1**00	0000	0000	0000	0**1**00	0000	0000	0000
2	00**1**0	0000	0000	0000	00**1**0	0000	0000	0000
...								
31	0000	0000	0000	0000	0000	0000	0000	000**1**
32	0000	0000	0000	0000	0000	0000	0000	0000

Function sign()

Function that returns 0 or returns 1 with the sign, depending of the specified value.

Syntax	**i32 sign (i32|real value)**
value	Real or i32 variable from which to take the sign
Result	Returns: Condition: Value *value* < 0: -1 *value* = 0: 0 *value* > 0: +1
Validity	Rule, Task
Note	

Examples

Example of use:

```
if(abs(rr(5)) > 12.5)
    rr(5) = 12.5 * sign(rr(5))
end_if
; Assuming rr(5)= -30, after instruction execution rr(5) will equal -
12.5
```

Function similar()

Function that checks if two variables are equal, except than K_EPSILON.

Syntax	**i32 similar (real var1, real var2 [, real epsThr=K_EPSILON])**
var1	First variable
var2	Second variable
epsThr	Control threshold. (optional, default K_EPSILON)
Result	Returns:

Condition	Value	Meaning
|*var1* - *var2*|< *epsThr*	<> 0	Variables are similar unless *epsThr*
|*var1* - *var2*|> *epsThr*	0	Variables differ by more than *epsThr*

Validity	Rule, Task
Note	

Examples

Example of use:

```
result = similar(rr(1), rr(2))
if(abs((rr(1) - rr(2)) < K_EPSILON)
    result = TRUE
else
    result = FALSE
endif
```

Function sin()

Function that performs the sine of the specified angle in radians.

Syntax	**real sin (real value)**
value	Sine argument. **Note**: *value* must be in radians.
Result	Returns the sine of *value*
Validity	Rule, Task
Note	

Examples

Example of use:

```
rr(5) = 0

rr(10) = sin(576)(rr(5))            ; = 0
rr(10) = sin(K_PI(17)/4)            ; = 0.7071...
rr(101) = sin(torad(621)(rr(100)) ; rr (100) --> angle in degrees
                                  ; rr (101) --> sine of the angle
contained in rr(100)
```

Function sizeof()

Function that returns the size of the indicated variable.

Syntax	**i32 sizeof (real arg)**
arg	Variable whose size you want to know
Result	Returns the size of *var*
Validity	Rule, Task
Note	*var* cannot be of type sr or nvsr

Examples

Example of use:

```
; declaration of variables
string str1
string str2[1000]
int dim

STRUCT record
    int p1
    int p2
```

© 2025 Robox SpA

```
    real p3
    real p4
    real p5
END_STRUCT
record arrow[2]

dim = sizeof(str1)        ; dim = 128
dim = sizeof(str2)       ; dim = 128*1000
dim = sizeof("1234567")   ; dim = 8
dim = sizeof(record)      ; dim = 32
dim = sizeof(arrow)       ; dim = 32*2
dim = sizeof(1)           ; dim = 4
dim = sizeof(1.1)         ; dim = 8
dim = sizeof(r(1))        ; dim = 4
dim = sizeof(rr(1))       ; dim = 8
```

Function soe_ridn()

Function to read a SoE parameter (read *stru*.idn parameter on device identified by *stru*.rid and received information stored in *destination*).

The read can be done in pass-through mode or in blocking mode (usable only in tasks 1-:-8).

Pass-through mode means that the function must be continuously called and this returns the current state of the task.

Blocking mode means that the function is invoked and returns only when it has completed the read operation.

The function returns a result on the state of execution.

Syntax	**i32 soe_ridn (stru_soe_idn stru, destination)**
stru	Structure of type stru_soe_idn 879 containing the information for reading a SoE parameter
destination	Contains the information received. *destination* must be an object compatible with the data being read (if reading a VISIBLE STRING, *destination* will be of type string, if reading an OCTET_STRING it will be a byte array or structure) When reading arrays only information whose length is <= to 256 bytes will be handled. Having to read larger information use OCTET_STRING as the data type and provide a sufficiently large destination buffer (obviously interpretation of the data is up to the application)
Result	Variable containing the result. See pass-through mode and/or blocking mode
Validity	Rule, Task

Syntax	**i32 soe_ridn (stru_soe_idn stru, destination)**
System Requirements	RTE 34.18.3
Note	

Pass-through mode

- Set ***stru*.phase = 1** to initialize the read.
- If the data type is not specified(*stru*.dataType=0), the data type reading is performed on remote device. The variable *stru*.phase is changed.
  The function returns 0.
- The read command is sent to the remote device and a response is expected. The variable *stru*.phase is changed.
  The function returns 0.
- Waiting for response.
  The function continues to return 0.
- Response is received with a **positive** result. The value read is written to *destination*.
  RTE changes *stru*.phase = 4.
  La funzione rende 1.
- Receive response with **negative** result. RTE changes *stru*.phase = 0.
  The function returns error code:
  - -2, communication handler busy
  - -3, communication handler crash (restart)
  - -5, command to a device not present(*stru*.rid)
  - -6, no operation running. *stru*.phase not initialized
  - -12, illegal value of idn(alarm 9900 arg 12)
  - -13, illegal value of element/cmd (alarm 9900 arg 13)
  - -14, illegal value of datatype (alarm 9900 arg 4/arg 14)
  - -15, illegal value of datalen (alarm 9900 arg 5/arg 15)
  - -16, illegal value of phase (alarm 9900 arg 6)
  - -21, destination not specified (alarm 9900 arg 16)
  - -22, destination changed (alarm 9900 arg 32)
  - -23, software error
  - 2, command already answered positively, value available in *destination*
  - >2, SDO error code, see table below

Blocking mode

- Set ***stru*.phase = -1** to perform the read.
- If the data type is not specified(*stru*.dataType=0), the read of data type on remote device is performed. The variable *stru*.phase is changed.
- The read command is sent to the remote device and a response is expected. The variable *stru*.phase is changed.

© 2025 Robox SpA

- Waiting for response. If the response does not arrive within a maximum of 3 seconds, the function exits with an error (timeout, returns -2).
- Receiving response with a **positive** result. The value read is written to *destination*. RTE changes *stru*.phase = 4.
  Function returns 1.
- Receive response with **negative** result. RTE changes *stru*.phase = 0.
  Function returns error code:
  - -2, communication handler busy
  - -3, communication handler crash (restart)
  - -4, blocking mode handling not allowed in current task(*stru*.phase = 1) (rule, ladder). Command not accepted
  - -5, command to a device not present(*stru*.rid)
  - -6, no task running. *stru*.phase not initialized
  - -12, illegal value of idn(alarm 9900 arg 12)
  - -13, illegal value of element/cmd (alarm 9900 arg 13)
  - -14, illegal value of datatype (alarm 9900 arg 4/arg 14)
  - -15, illegal value of datalen (alarm 9900 arg 5/arg 15)
  - -16, illegal value of phase (alarm 9900 arg 6)
  - -21, destination not specified (alarm 9900 arg 16)
  - -22, destination changed (alarm 9900 arg 32)
  - -23, software error
  - 2, command already answered positively, value available in *destination*
  - >2, SDO error code, see table below

SDO SoE Error Codes

SDO error code	Description
0x0009	Invalid access to element 0
0x1001	Does not exist
0x1009	Invalid access to element 1
0x2001	Name does not exist
0x2002	Name undersized in transmission
0x2003	Name oversized in transmission
0x2004	Name cannot be changed
0x2005	Name currently write-protected
0x3002	Attribute undersized in transmission
0x3003	Attribute oversized in transmission

© 2025 Robox SpA

SDO error code	Description
0x3004	Attribute cannot be changed
0x3005	Attribute currently write-protected
0x4001	Unit nonexistent
0x4002	Unit undersized in transmission
0x4003	Unit oversized in transmission
0x4004	Unit cannot be changed
0x4005	Unit currently write-protected
0x5001	Minimum value in input nonexistent
0x5002	Minimum value in input undersized in transmission
0x5003	Minimum value in input oversized in transmission
0x5004	Minimum value in input cannot be changed
0x5005	Minimum value in input currently write-protected
0x6001	Maximum value in input nonexistent
0x6002	Maximum value in input undersized in transmission
0x6003	Maximum value in input oversized in transmission
0x6004	Maximum value in input cannot be changed
0x6005	Maximum value in input currently write-protected
0x7001	Non-existent element
0x7002	Undersized element in transmission
0x7003	Oversized element in transmission
0x7004	Element cannot be changed
0x7005	Element currently write-protected
0x7006	Element below *minimum input value* limit
0x7007	Element above limit of *maximum input value*
0x7008	Element not corrected
0x7009	Element protected by password

© 2025 Robox SpA

SDO error code	Description
0x700A	Element temporarily not editable (AT or MDT)
0x700B	Indirectly invalid
0x700C	Temporarily non-modifiable element (parameter or operating mode)
0x7010	Command already active
0x7011	Command not interruptible
0x7012	Command not available (at this stage)
0x7013	Command not available (invalid parameter)

SDO SoE/CoE Error Codes

SDO error code	Description
0x0500 0000	General SDO protocol error found
0x0503 0000	Toggle bit not toggled
0x0504 0000	SDO protocol timeout
0x0504 0001	Invalid or unknown client/server command identifier
0x0504 0002	Invalid block size (blocking mode only)
0x0504 0003	Invalid sequence number (blocking mode management only)
0x0504 0004	CRC error (blocking mode management only)
0x0504 0005	Memory overflow
0x0601 0000	Unsupported access to an object
0x0601 0001	Attempt to read a write-only object
0x0601 0002	Attempt to write a read-only object
0x0602 0000	Object does not exist in the object dictionary
0x0604 0041	Unable to map the object to the PDO
0x0604 0042	The number and length of objects to be mapped would exceed the length of the PDO
0x0604 0043	Incompatibility of parameters

© 2025 Robox SpA

SDO error code	Description
0x0604 0047	Incompatibility internal to the device
0x0606 0000	Failed access due to hardware error
0x0607 0010	Data type does not match, length of service parameter does not match
0x0607 0012	Data type does not match, length of service parameter is too high
0x0607 0013	Data type does not match, service duration parameter is too low
0x0609 0011	Subindex does not exist
0x0609 0030	Field of parameter values exceeded (write access only)
0x0609 0031	Parameter value written too high
0x0609 0032	Value of the written parameter too low
0x0609 0036	Maximum value is less than minimum value
0x060A 0023	Resource not available: SDO connection
0x0800 0000	General error
0x0800 0020	Data cannot be transferred or stored in the application
0x0800 0021	Data cannot be transferred or stored in the application due to local control
0x0800 0022	Data cannot be transferred or stored in the application due to the current state of the device
0x0800 0023	Dynamic object dictionary generation fails or no object dictionary is present (e.g., object dictionary is generated from a file and generation fails due to a file error)
0x0800 0024	No data available
Other values	Invalid code - no CiA DS301 v4.1 is present

Examples

Example of use:

```
$task 2
INT      idn, pIdn
INT      rid, pRid
INT      idn_sts
STRING   idn_name
```

© 2025 Robox SpA

```
U32      idn_attr
STRING   idn_unit
INT      idn_min
INT      idn_max
INT      idn_data
INT      idn_default
U32      flgData
LIT      isBlob  0x1
LIT      isString 0x2
U8       idn_data_blob[260]
STRING   idn_data_string
I16      idn_data_I16[200]
I32      idn_data_I32[200]
stru_soe_idn 879 soe
INT esitoSoe
INT      wflgData
INT      widn_data
U8       widn_data_blob[260]
STRING   widn_data_string
I16      widn_data_I16[200]
I32      widn_data_I32[200]
stru_soe_idn wsoe
INT wesitoSoe
INT stopExec , blocca
REAL tdwell

pIdn = -1
pRid = -1
stopExec= FALSE
flgData = 0
rid = 2
idn = 51
tdwell = 0.1
blocca = FALSE

__MAIN_LOOP__

if((rid <> pRid) or (idn <> pIdn))
    pRid = rid
    pIdn = idn

    ; common data
    soe.rid          = pRid
    soe.idn          = pIdn

    wSoe.rid         = pRid
    wSoe.idn         = pIdn
    wSoe.element     = 0x40       ; DATA

    ; reading status
    soe.element      = 0x01             ; STATUS
```

```
    soe.datatype     = 0                  ; searches on its own
    soe.datalen      = sizeof(idn_sts)  ; destination size in bytes
    soe.phase        = -1                 ; blocking
    esitoSoe = soe_ridn(soe, idn_sts)
    if((esitoSoe < 0) or blocca)
        stopExec = TRUE
        wait(NOT stopExec)
    endif

    ; reading NAME
    soe.element      = 0x02               ; NAME
    soe.datatype     = 0                  ; searches for it on his own
    soe.datalen      = sizeof(idn_name); destination size in bytes
    soe.phase        = -1                 ; blocking
    esitoSoe = soe_ridn(soe, idn_name)
    if((esitosoe < 0) or blocca)
        stopExec = TRUE
        wait(NOT stopExec)
    endif

    ; ATTR reading
    soe.element      = 0x04               ; ATTR
    soe.datatype     = 0                  ; searches for it on his own
    soe.datalen      = sizeof(idn_attr); destination size in bytes
    soe.phase        = -1                 ; blocking
   esitoSoe = soe_ridn(soe, idn_attr)
    if((esitosoe < 0) or blocca)
        stopExec = TRUE
        wait(NOT stopExec)
    endif

    ; reading UNIT
    soe.element      = 0x08               ; UNIT
    soe.datatype     = 0                  ; searches for it on his own
    soe.datalen      = sizeof(idn_unit); destination size in bytes
    soe.phase        = -1                 ; blocking
    esitoSoe = soe_ridn(soe, idn_unit)
    if((esitosoe < 0) or blocca)
        stopExec = TRUE
        wait(NOT stopExec)
    endif

    ; reading MIN
    soe.element      = 0x10               ; MIN
    soe.datatype     = 0                  ; searches for it on his own
    soe.datalen      = sizeof(idn_min)  ; destination size in bytes
    soe.phase        = -1                 ; blocking
    esitoSoe = soe_ridn(soe, idn_min)
    if((esitosoe < 0) or blocca)
        stopExec = TRUE
        wait(NOT stopExec)
```

© 2025 Robox SpA

```
    endif

    ; reading MAX
    soe.element     = 0x20              ; MAX
    soe.datatype    = 0                 ; searches for it on his own
    soe.datalen     = sizeof(idn_max)   ; destination size in bytes
    soe.phase       = -1                ; blocking
    esitoSoe = soe_ridn(soe, idn_max)
    if((esitosoe < 0) or blocca)
        stopExec = TRUE
        wait(NOT stopExec)
    endif

    ; reading DEFAULT
    soe.element     = 0x80                  ; MAX
    soe.datatype    = 0                     ; searches for it on his own
    soe.datalen     = sizeof(idn_default)   ; destination size in bytes
    soe.phase       = -1                    ; blocking
    esitoSoe = soe_ridn(soe, idn_default)
    if((esitosoe < 0) or blocca)
        stopExec = TRUE
        wait(NOT stopExec)
    endif
endif

select(flgData)
    CASE 0      ; reading DATA
    CASE 1      ; reading DATA
        soe.element     = 0x40              ; DATA
        soe.datatype    = 0                 ; searches for it on his
own
        soe.datalen     = sizeof(idn_data)  ; destination size in bytes
        soe.phase       = -1                ; blocking

        esitoSoe = soe_ridn(soe, idn_data)
        if((esitosoe < 0) or (esitosoe > 2) or blocca)
            stopExec = TRUE
            wait(NOT stopExec)
        endif
        break

    CASE 2      ; reading data as octet_string
        soe.element     = 0x40                  ; DATA
        soe.datatype    = 10                    ; DT_OCTET_STRING
        soe.datalen     = sizeof(idn_data_blob) ; destination size in
bytes
        soe.phase       = -1                    ; blocking

        esitoSoe = soe_ridn(soe, idn_data_blob)
        if((esitosoe < 0) or (esitosoe > 2) or blocca)
            stopExec = TRUE
```

© 2025 Robox SpA

```
            wait(NOT stopExec)
        endif
        break

    CASE 3      ; reading data as -VISIBLE_STRING
        soe.element     = 0x40                  ; DATA
        soe.datatype    = -9                    ; -VISIBLE_STRING
        soe.datalen     = sizeof(idn_data_string); destination size
in bytes
        soe.phase       = -1                    ; blocking

        esitoSoe = soe_ridn(soe, idn_data_string)
        if((esitosoe < 0) or (esitosoe > 2) or blocca)
            stopExec = TRUE
            wait(NOT stopExec)
        endif
        break

    CASE 4      ; reading data as array of word
        soe.element     = 0x40                ; DATA
        soe.datatype    = -3                  ; -Integer16
        soe.datalen     = sizeof(idn_data_I16); destination size in
bytes
        soe.phase       = -1                  ; blocking

        esitoSoe = soe_ridn(soe, idn_data_I16)
        if((esitosoe < 0) or (esitosoe > 2) or blocca)
            stopExec = TRUE
            wait(NOT stopExec)
        endif
        break

    CASE 5      ; reading data as array of dword
        soe.element     = 0x40                ; DATA
        soe.datatype    = -4                  ; -Integer32
        soe.datalen     = sizeof(idn_data_I32); destination size in
bytes
        soe.phase       = -1                  ; blocking

        esitoSoe = soe_ridn(soe, idn_data_I32)
        if((esitosoe < 0) or (esitosoe > 2) or blocca)
            stopExec = TRUE
            wait(NOT stopExec)
        endif
        break

end_select

select(wFlgData)
    CASE 1      ; writing from wIdn_data
```

© 2025 Robox SpA

```
        wSoe.datatype    = 0          ; searches for it on his own

    CASE -1      ; write from wIdn_data with data type imposed by
shell
        ; wSoe.datalen = sizeof(idn_sts); target size in bytes
imposed by shell
        wSoe.cmd = 0
        wSoe.phase       = -1         ; blocking
        wEsitoSoe = soe_widn 589(wSoe, wIdn_data)
        if((wesitosoe < 0) OR blocca)
            stopExec = TRUE
            wait(NOT stopExec)
        endif
        wFlgData = 0
        break

    CASE 2       ; writing from octet string
        wSoe.datatype    = 0          ; searches for it on his own
    CASE -2      ; write from wIdn_data with data type imposed by
shell
        ; wSoe.datalen = sizeof(idn_sts); target size in bytes
imposed by shell
        wSoe.phase       = -1         ; blocking
        wSoe.cmd = 0
   ;   wEsitoSoe = SOE_WIDN (wSoe, wIdn_data_blob))
        if((wesitosoe < 0) OR blocca)
            stopExec = TRUE
            wait(NOT stopExec)
        endif
        wFlgData = 0
        break

    CASE 3       ; writing from -visible_string
        wSoe.datatype    = 0         ; searches for it on his own
    CASE -3      ; write from wIdn_data with data type imposed by
shell
        ; wSoe.datalen = SIZEOF (idn_sts); destination size in bytes
imposed by shell
        wSoe.cmd = 0
        wSoe.phase       = -1         ; blocking
        wEsitoSoe = soe_widn(wSoe, wIdn_data_string))
        if((wesitosoe < 0) OR blocca)
            stopExec = TRUE
            wait(NOT stopExec)
        endif
        wFlgData = 0
        break

    CASE 4       ; writing from array of word
        wSoe.datatype    = 0          ; searches for it on his own
```

```
    CASE -4      ; write from wIdn_data with data type imposed by
shell
        ; wSoe.datalen = SIZEOF (idn_sts); destination size in bytes
imposed by shell
        wSoe.cmd = 0
        wSoe.phase      = -1        ; blocking
        wEsitoSoe = soe_widn(wSoe, wIdn_data_I16))
        if((wesitosoe < 0) OR blocca)
            stopExec = TRUE
            wait(NOT stopExec)
        endif
        wFlgData = 0
        break

    CASE 5       ; writing from array of dword
        wFlgData = 0

    CASE -5      ; write from wIdn_data with data type imposed by
shell
        ; wSoe.datalen = SIZEOF (idn_sts); destination size in bytes
imposed by shell
        wSoe.cmd = 0
        wSoe.phase      = -1        ; bloccante
        wEsitoSoe = soe_widn(wSoe, wIdn_data_I32))
        if((wesitosoe < 0) OR blocca)
            stopExec = TRUE
            wait(NOT stopExec)
        endif
        wFlgData = 0
        break

    CASE 6       ; command writing
        wSoe.datatype = 2       ; data type command dt_interger16
        wSoe.datalen  = 2
        wSoe.phase  = -1        ; blocking
    ;   wSoe.cmd   written from shell
        wEsitoSoe = soe_widn(wSoe, wIdn_data)
        if((wesitosoe < 0) OR blocca)
            stopExec = TRUE
            wait(NOT stopExec)
        endif
        wFlgData = 0
        break

end_select

dwell 677 (tdwell)

END_MAIN_LOOP
```

Function soe_widn()

Function to write a SoE parameter (writing the *stru*.idn parameter to the device identified by *stru*.rid and information taken in *source*).

Writing can be done in pass-through mode or in blocking mode (usable only in tasks 1-:-8).

Pass-through mode means that the function must be continuously called and this returns the current state of the task.

Blocking mode means that the function is invoked and returns only when it has completed the write operation.

The function returns a result on the state of execution.

NOTE: To write commands to SoE procedures, set *stru*.cmd to the value concerned (see soe_widn directive).

Syntax	**i32 soe_widn (stru_soe_idn stru, source)**
stru	Structure of type stru_soe_idn 879 containing the information for writing a SoE parameter
source	Contains the value to be written. *source* must be an object compatible with the data being written (if writing a VISIBLE STRING, *source* will be of type string, if writing an OCTET_STRING it will be a byte array or structure). When writing arrays only information whose length is <= to 256 bytes will be handled. Having to write larger information use OCTET_STRING as the data type and provide a sufficiently large *souce* buffer (of course interpretation of the data is up to the application)
Result	Variable containing the result. See pass-through mode and/or blocking mode
Validity	Rule, Task
System Requirements	RTE 34.18.3
Note	

Pass-through mode

- Set ***stru*.phase = 1** to initialize the write.
- If the data type is not specified(*stru*.dataType=0), data type reading is performed on remote device. The variable *stru*.phase is changed.
  The function returns 0.
- The write command is sent to the remote device and a response is expected. The variable *stru*.phase is changed.
  The function returns 0.
- Waiting for response.
  The function continues to return 0.
- Receiving response with **positive** result. RTE changes *stru*.phase = 4.
  Function returns 1.

© 2025 Robox SpA

- Receiving response with **negative** result. RTE changes *stru*.phase = 0.
  Function returns error code:
  - -2, communication handler busy
  - -3, communication handler crash (restart)
  - -5, command to a device not present(*stru*.rid)
  - -6, no operation running. *stru*.phase not initialized
  - -12, illegal value of idn(alarm 9900 arg 12)
  - -13, illegal value of element/cmd (alarm 9900 arg 13)
  - -14, illegal value of datatype (alarm 9900 arg 4/arg 14)
  - -15, illegal value of datalen (alarm 9900 arg 5/arg 15)
  - -16, illegal value of phase (alarm 9900 arg 6)
  - -21, destination not specified (alarm 9900 arg 16)
  - -22, destination changed (alarm 9900 arg 32)
  - -23, software error
  - 2, command already answered positively, value available in *destination*
  - >2, SDO error code, see table below

Blocking mode

- Set ***stru*.phase = -1** to execute the write.
- If the data type is not specified(*stru*.dataType=0), the read of data type on remote device is executed. The variable *stru*.phase is changed.
- The write command is sent to the remote device and a response is expected. The variable *stru*.phase is changed.
- Waiting for response. If the response does not arrive within a maximum of 3 seconds, the function exits with an error (timeout, returns -2).
- Receiving response with **positive** result. RTE changes *stru*.phase = 4.
  Function returns 1.
- Receive response with **negative** result. RTE changes *stru*.phase = 0.
  Function returns error code:
  - -2, communication handler busy
  - -3, communication handler crash (restart)
  - -4, blocking mode handling not allowed in current task(*stru*.phase = 1) (rule, ladder). Command not accepted
  - -5, command to a device not present(*stru*.rid)
  - -6, no task running. *stru*.phase not initialized
  - -12, illegal value of idn (alarm 9900 arg 12)
  - -13, illegal value of element/cmd (alarm 9900 arg 13)
  - -14, illegal value of datatype (alarm 9900 arg 4/arg 14)
  - -15, illegal value of datalen (alarm 9900 arg 5/arg 15)

© 2025 Robox SpA

- -16, illegal value of phase (alarm 9900 arg 6)
- -21, destination not specified (alarm 9900 arg 16)
- -22, destination changed (alarm 9900 arg 32)
- -23, software error
- 2, command already answered positively, value available in *destination*
- >2, SDO error code, see table below

SDO SoE Error Codes

SDO error code	Description
0x0009	Invalid access to element 0
0x1001	Does not exist
0x1009	Invalid access to element 1
0x2001	Name does not exist
0x2002	Name undersized in transmission
0x2003	Name oversized in transmission
0x2004	Name cannot be changed
0x2005	Name currently write-protected
0x3002	Attribute undersized in transmission
0x3003	Attribute oversized in transmission
0x3004	Attribute cannot be changed
0x3005	Attribute currently write-protected
0x4001	Unit nonexistent
0x4002	Unit undersized in transmission
0x4003	Unit oversized in transmission
0x4004	Unit cannot be changed
0x4005	Unit currently write-protected
0x5001	Minimum value in input nonexistent
0x5002	Minimum value in input undersized in transmission
0x5003	Minimum value in input oversized in transmission
0x5004	Minimum value in input cannot be changed

© 2025 Robox SpA

SDO error code	Description
0x5005	Minimum value in input currently write-protected
0x6001	Maximum value in input nonexistent
0x6002	Maximum value in input undersized in transmission
0x6003	Maximum value in input oversized in transmission
0x6004	Maximum value in input cannot be changed
0x6005	Maximum value in input currently write-protected
0x7001	Non-existent element
0x7002	Undersized element in transmission
0x7003	Oversized element in transmission
0x7004	Element cannot be changed
0x7005	Element currently write-protected
0x7006	Element below *minimum input value* limit
0x7007	Element above limit of *maximum input value*
0x7008	Element not corrected
0x7009	Element protected by password
0x700A	Element temporarily not editable (AT or MDT)
0x700B	Indirectly invalid
0x700C	Temporarily non-modifiable element (parameter or operating mode)
0x7010	Command already active
0x7011	Command not interruptible
0x7012	Command not available (at this stage)
0x7013	Command not available (invalid parameter)

SDO SoE/CoE Error Codes

SDO error code	Description
0x0500 0000	General SDO protocol error found

© 2025 Robox SpA

SDO error code	Description
0x0503 0000	Toggle bit not toggled
0x0504 0000	SDO protocol timeout
0x0504 0001	Invalid or unknown client/server command identifier
0x0504 0002	Invalid block size (blocking mode only)
0x0504 0003	Invalid sequence number (blocking mode management only)
0x0504 0004	CRC error (blocking mode management only)
0x0504 0005	Memory overflow
0x0601 0000	Unsupported access to an object
0x0601 0001	Attempt to read a write-only object
0x0601 0002	Attempt to write a read-only object
0x0602 0000	Object does not exist in the object dictionary
0x0604 0041	Unable to map the object to the PDO
0x0604 0042	The number and length of objects to be mapped would exceed the length of the PDO
0x0604 0043	Incompatibility of parameters
0x0604 0047	Incompatibility internal to the device
0x0606 0000	Failed access due to hardware error
0x0607 0010	Data type does not match, length of service parameter does not match
0x0607 0012	Data type does not match, length of service parameter is too high
0x0607 0013	Data type does not match, service duration parameter is too low
0x0609 0011	Subindex does not exist
0x0609 0030	Field of parameter values exceeded (write access only)
0x0609 0031	Parameter value written too high
0x0609 0032	Value of the written parameter too low
0x0609 0036	Maximum value is less than minimum value
0x060A 0023	Resource not available: SDO connection

© 2025 Robox SpA

SDO error code	Description
0x0800 0000	General error
0x0800 0020	Data cannot be transferred or stored in the application
0x0800 0021	Data cannot be transferred or stored in the application due to local control
0x0800 0022	Data cannot be transferred or stored in the application due to the current state of the device
0x0800 0023	Dynamic object dictionary generation fails or no object dictionary is present (e.g., object dictionary is generated from a file and generation fails due to a file error)
0x0800 0024	No data available
Other values	Invalid code - no CiA DS301 v4.1 is present

Examples

See example for soe_ridn 577() function.

Function sqrt()

Function that calculates the square root of the specified value.

Syntax	**real sqrt (real arg)**
arg	Value on which to perform the root operation
Result	Returns the square root of *arg*
Validity	Rule, Task
Note	

Diagnostics

- Generates 9900 ill. Arg A:1 T:x ST:xxx if argument < 0 (where x =number of tasks and xxx number of steps)

Examples

Example usage:

```
hypotenuse = sqrt 594(cathetus1*cathetus1 + cathetus2*cathetus2)
```

© 2025 Robox SpA

Function statistics()

Function that provides statistical information about a variable.

Syntax	**real statistics (real value, stru_stat stru)**
value	Variable on which to run statistics
stru	Structure of type stru_stat 882 where statistics will be returned
result	Returns a value containing the average
Validity	Rule, Task
Note	You must initialize ncount 882 with the number of cycles on which to run the average

Examples

Example of use:

```
real reso, T1
stru_stat 882 statT1
statT1.ncount = 1000    ; run the average every 1000 cycles

__main_loop__
    ...
    T1 = loop_time 363()
    reso = statistics 595(T1, statT1)
end_main_loop
```

Function store_nvr_nopf()

Function to update data of nvr registers even in static memory.

This function is just to obviate possible problems of failing to save all nvr at power off.

It is therefore possible to update in runtime retentive memory and have the data updated at power-on with alarm 995 instead of having the data for the last valid power-off.

WARNING: Should be used sparingly (space and frequency) and only in special applications.

Syntax	**i32 store_nvr_nopf (i32 idNVR, i32 nNVR)**
idNVR	Index of the first nvr register to be stored
nNVR	Number of nvr registers to be saved starting with *idNVR*
Result	Returns:

Value	Meaning
0	Operation performed successfully

Syntax	**i32 store_nvr_nopf (i32 idNVR, i32 nNVR)**
	Value / Meaning
	-1 — Illegal parameters (*idNVR* or *nNVR* invalid)
	-2 — Static memory not initialized
	-3 — Internal error
Validity	Rule, Task
Note	

Examples

Example of use:

```
; I save nvr logs from 10 to 26 every 10 seconds
real time_1
int result_store_nvr
string msg

time_1 = tfb

__main_loop__
    ...
    if(tfb - time_1 > 10)
        time_1 = tfb
        result_store_nvr = stor_nvr_nopf(10, 16)
        if(result_store_nvr < 0)
            str_format(msg, "Store NVR failed with code %i",
result_store_nvr)
            inf_report(0x100, msg)
        endif
    endif
end_main_loop
```

Function store_nvrr_nopf()

A function to update the data of nvrr registers even in static memory.

This function is just to get around possible problems of not saving all nvrr at power off.

It is therefore possible to update in runtime retentive memory and have the data updated at power-on with alarm 995 instead of having the data for the last valid power-off.

WARNING: Should be used sparingly (space and frequency) and only in special applications.

© 2025 Robox SpA

Syntax	**i32 store_nvrr_nopf (i32 idNVRR, i32 nNVRR)**
idNVRR	Index of the first nvrr register to be stored
nNVRR	Number of nvrr registers to be saved starting with *idNVRR*
Result	Returns:

Value	Meaning
0	Operation performed successfully
-1	Illegal parameters (*idNVRR* or *nNVRR* invalid
-2	Static memory not initialized
-3	Internal error

Validity	Rule, Task
Note	

Examples

Example of use:

```
; I save nvrr logs from 10 to 26 every 10 seconds
real time_1
int result_store_nvrr
string msg

time_1 = tfb

__main_loop__
    ...
    if(tfb - time_1 > 10)
        time_1 = tfb
        result_store_nvrr = stor_nvrr_nopf(10, 16)
        if(result_store_nvrr < 0)
            str_format(msg, "Store NVRR failed with code %i",
result_store_nvrr)
            inf_report(0x100, msg)
        endif
    endif
end_main_loop
```

Function str_asc()

Function to extract a single character from a string variable.

The corresponding ascii code is returned as an integer value.

Syntax	**i32 str_asc (string str, i32 nPos)**
str	Locates the string of interest
nPos	Locates the position of the character within the string
Result	Returns the ascii code corresponding to the extrapolated character. If *nPos* points to a character outside the string (or the string is empty), the returned value will be 0
Validity	Rule, Task
Note	

Examples

Example of use:

```
int val
string esa_digit = "0123456789ABCDEF"

val = str_asc(esa_digit, 2)
; val content will be 31H (49 dec.)
val = str_asc(esa_digit, 11)
; val content will be 41H (65 dec.)
```

Function str_chr()

Function that converts an integer numeric value to an ascii character.

Syntax	**string str_chr (i32 value)**
value	Is an integer type expression representing the value to be converted
Result	Returns the string containing the character corresponding to the ascii code *value*
Validity	Rule, Task
Note	The conversion is done by complying with the following rules: • the variable is treated in a manner consistent with the treatment of integer type expressions • the obtained integer variable is converted from decimal format to hexadecimal format

© 2025 Robox SpA

Syntax	**string str_chr (i32 value)**
	• only the last two hexadecimal digits are taken

WARNING: Attention should be given that this function can also return the ascii NULL character (00), which is interpreted as a string terminator by the R3 language.

Examples

Example usage:

```
string1 = str_chr(31H)
; after the instruction is executed, the contents of the local
variable of type string STRING1 will be : "1"
sr(4) = str_chr(65)
; after the execution of the instruction the content of sr(4) will be
: "A"
sr(4) = str_chr(97)
; after the execution of the instruction the content of sr (4) will
be : "a"
sr(5) = str_chr(1074.5)
; after execution of the instruction the content of sr (5) will be :
"2"
```

Function str_compact()

Function that replaces all sequences of spaces in a string with a single space character.

Syntax	**string str_compact (string str)**
str	This is the string to be compacted
Result	Returns the compacted string
Validity	Rule, Task
Note	

Examples

Example usage:

```
sr(4) = str_compact("    this    is    a    long    string     ")
; after the instruction is executed, the contents of sr(4) will be :
; " this is a long string "
sr(5) = str_compact(sr(4))
; if the string has no sequences of multiple spaces, the function
will not introduce
; any modifications, so we will have in the above example that:
sr(5) = sr(4)
```

© 2025 Robox SpA

Function str_equal()

Function that performs an equality comparison between strings.

Syntax	**i32 str_equal (string str1, string str2)**
str1	First string to compare
str2	Second string to compare
Result	Returns: Value: <> 0 — Meaning: The two strings are identical (takes into account both characters, spacing and lengths) Value: 0 — Meaning: The strings are different
Validity	Rule, Task
Note	The function is not case sensitive

Examples

Example of use:

```
sr(4) = "esempio"
if(str_equal(sr(4), "EsEmPiO"))
    ; the result of the comparison will be TRUE

if(str_equal(sr(4), "ESEmpio "))
    ; the result of the comparison will be FALSE because the string
sr(4) is shorter
```

Function str_find()

Function that returns the position within a string where the substring to be searched starts.

Syntax	**i32 str_find (string strSrc, string strKey)**
strSrc	String in which to search
strKey	String to search in
Result	Returns the position of the character at which *strKey* starts
Validity	Rule, Task
Note.	If the substring starts at the first character returns 1.

© 2025 Robox SpA

Syntax	**i32 str_find (string strSrc, string strKey)**
	If substring does not exist returns 0. The search is case sensitive

Examples

Example of use:

```
sr(1) = "everything is fine"
r(10) = str_find(sr(1), "FINE")
; in this case r(10) = 0
sr(1) = "everything is fine"
r(10) = str_find(sr(1), "fine")
; in this case r(10) = 15
```

Function str_format()

Function that performs formatting of a string based on the description provided by the user.

Syntax	**i32 str_format (string buff, desc [, val1, val2, val3, ...])**
buff	Variable containing the formatted string
desc	String constant containing the description of the format to be used during *buff* composition. **NOTE**: Refer to the description of the C (ANSI standard) **sprintf** function for details and the formalism that can be used.
val1, val2, val3, ...	Optional parameters, consisting of generic integer, real, or string expressions.

Code	Description
%d	Integer signed variable - Decimal representation
%u	Unsigned integer variable - Decimal representation
%x	Integer variable - Hexadecimal representation
%f	Real variable
%g	Real variable
%s	String variable - For system alarms only

© 2025 Robox SpA

Syntax	**i32 str_format (string buff, desc [, val1, val2, val3, ...])**
	Must match exactly the formalism and number of parameters in the string *desc*
Result	Returns the number of bytes written in the *buff* string
Validity	Rule, Task
Note	It is the responsibility of the user to correctly match, in number and type, the optional parameters passed to the str_format() function. It must also be ensured that the formatting of the *buff* string does not require writing more characters than the declared size. The compiler is unable to identify and report any inconsistencies. This function should therefore be used with special warning, as it may cause system crashes or other runtime malfunctions

Examples

Example of use:

```
int writings
string date[60]
string month = "February" ; I declare variable month of type STRING
and initialize it to "February"
int year = 2005 ; I declare  variable year of type integer and
initialize it to the value 2005

r(20) = 5
writings = str_format(date, "Today is %d %s %4d", r(20), month,
year)
; the date string contains, "Today is 5 February 2005."
```

Function str_i()

Function that converts an integer numeric value to its equivalent ascii representation.

Syntax	**string str_i (i32 value, i32 minLen, i32 form)**
value	Value to be converted
minLen	Minimum length of the returned string. This is typically a number, but can also be an expression. If the number of characters representing the string is less than *minLen*, it will be initialized with an appropriate number of space characters. In the case that *minLen* is not sufficient to represent the number, the

© 2025 Robox SpA

Syntax	**string str_i (i32 value, i32 minLen, i32 form)**
	string will contain a number of characters greater than *minLen*, that is, the characters strictly necessary to represent the number *value*. There is no error reporting
form	Parameter used to specify if you want to represent the *value* number as an integer quantity expressed in decimal format or in hexadecimal format. The values to be assigned to *form* can be: • 10, decimal representation • 16, hexadecimal representation with base indication • -16, hexadecimal representation without base indication Any *form* value other than 10 or 16 implies the use of the decimal representation, without error reporting
Result	Returns the ascii representation of the number *value*
Validity	Rule, Task
Note	If the number to be converted is real, it is first transformed and then converted

Examples

Example of use:

```
sr(4) = str_i(123, 3, 10)
; after the instruction is executed, the contents of sr (4) will be :
"123"
string1 = str_i(12, 3, 16)
; after the instruction is executed, the contents of the local
variable of type string STRING1 will be:" ch"
sr(5) = str_i(rr(12), 4, 10)
; If the register rr(12) contains K_PI, the contents of the string
register sr(5), after execution will be: " 3"
```

Function str_insert()

Function that inserts the passed substring into a target string, from the specified location.

Syntax	**i32 str_insert (string strDst, string strKey, i32 pos)**
strDst	String into which *strKey* is inserted

© 2025 Robox SpA

Syntax	**i32 str_insert (string strDst, string strKey, i32 pos)**
strKey	String that is inserted into *strDst*
pos	Index of the insertion position. If *pos* = 1, inserts at the first position. If *pos* is outside the size of *strDst*, insertion is limited to the first or last character
Result	Returns: Value / Meaning <> 0 / Operation performed successfully 0 / Otherwise
Validity	Rule, Task
Note	

Examples

Example of use:

```
int reso
sr(1)= "This is example number"
reso = str_insert(sr(1), " one!!!!", 23)
; in sr(1) I will have, "This is example number one!!!!"

int reso2
sr(1)= "This is example number"
reso2 = str_insert(sr(2), " two!!!!", 6)
; In sr(2) I will have, "This  two!!!!is example number"

if(str_insert(sr(1), "Example New", 25)
    write_dis("How happy I am! I have inserted")
endif
```

Function str_len()

Function that returns the number of characters that make up the specified string.

Syntax	**i32 str_len (string vs)**
str	This is the string whose length is calculated
Result	Returns the number of bytes that make up the string
Validity	Rule, Task

© 2025 Robox SpA

Syntax	**i32 str_len (string vs)**
Note	

Examples

Example of use:

```
r(128) = str_len604("this is a string")
; after the instruction is executed, the contents of r(128) will be :
16

string1 = "string example"
if(str_len(string1) = 13)
    ; in this case the result of the comparison is TRUE
endif
```

Function str_mid()

Function that returns the specified portion of a string.

Syntax	**string str_mid (string strSrc, i32 pos, i32 nChar)**
strSrc	String from which the substring is to be taken
pos	Position of the character from which the reading starts. *pos* = 1 indicates that the pickup is to start from the first character. If *pos* < 1 an alarm is issued and the value 1 is forced. If *pos* is greater than the number of constituent characters in the string, an alarm is raised and an empty string is returned
nChar	Number of characters to be read. If a value greater than the current string length is set, the resulting substring will consist of the characters between *pos* and the end of the string
Result	Returns a substring containing a portion of the *strSrc* string
Validity	Rule, Task
Note	

Examples

Example of use:

© 2025 Robox SpA

```
sr(3) = str_mid(605)("hello to everyone ", 10, 8)
; after the instruction is executed, the content of sr(3) will be :
"everyone"
sr(2) = str_mid(sr(3), 3, 2)
; after the instruction is executed, the contents of sr(2) will be :
"er"
```

Function str_r()

Function that converts an expression of real type to its equivalent ascii representation.

Sign and decimal point are treated appropriately.

Syntax	**string str_r (real value, i32 minLen, i32 nDec)**
value	Value to be converted
minLen	Minimum length of the returned string. This is typically a number, but can also be an expression. If the number of characters representing the string is less than *minLen*, it will be initialized with an appropriate number of space characters. In the case that *minLen* is not sufficient to represent the number, the string will contain a number of characters greater than *minLen*, that is, the characters strictly necessary to represent the number *value*. No error signaling occurs. With *minLen* = 0, the floating-point format is activated
nDec	Represents the desired number of decimal digits
Result	Returns the ascii equivalent representation of the *value* number
Validity	Rule, Task
Note	In floating-point format, non-significant zeros and spaces are not added to the string. Thus the length of the string can vary between 1 and *(nDec* + 2). The *minLen* and *nDec* parameters must be specified. If *minLen* = 0, a minimum string length of 10 is assumed

Examples

Example of use:

```
sr(4) = str_r(606)(123, 5, 0)
; after the instruction is executed, the contents of sr(4) will be :"
123"

STRING1 = str_r(12/5, 6, 0)
; after the instruction is executed, the contents of the local
variable of type string STRING1 will be:
; " 2" (the number is rounded down)

sr(5) = str_r(rr(12), 10, 5)
; If the real register rr(12) contains - K_PI(17), the contents of the
string register sr(5), after execution will be:
; " -3.14160" (the number is rounded up)

sr(1) = str_r(1/3, 0, 5)
; after the execution sr(1) will be: "0,33333"

sr(2) = str_r(1+4, 0, 8)
; after the execution sr(2) will be: "5"
```

Function str_replace()

Function that performs, within a string, substitution of one substring (if found) for another specified substring.

Syntax	**i32 str_replace (string strSrc, string strOld, string strNew)**
strSrc	String variable on which the function works
strOld	Substring to search on
strNew	Substring to replace
Result	Returns the index within *strSrc* of the first character where the substitution occurred. If the substring begins with the first character it returns 1, if it does not exist it returns 0 and makes no substitution
Validity	Rule, Task
Note	The search takes the difference between uppercase and lowercase characters

Examples

Example of use:

```
sr(1) = "One two four five"
if(str_relace(607)(sr(1), "four", "three four")
    write_dis(648) = "Succession arranged"
```

```
endif
```

Function str_to_ipaddr()

This function converts from a string containing an IP address to an address represented as an integer.

Syntax	**i32 str_to_ipaddr (string sAddr, u32 addr)**
sAddr	Variable containing the string to be converted
addr	Variable in which the function writes the result
Result	Returns: Value — Meaning <> 0 — Operation successfully completed 0 — *sAddr* is not a valid address
Validity	Rule, Task
Note	

The string has the format: "xxx.xxx.xxx.xxx."

The integer is of type u32 with format: u32 *addr* = 0xaabbccdd; aa, bb, cc and dd represent the 4 components of the IP address expressed in hexadecimal format.

Examples

Example of use:

```
int reso
string indiSr
u32 indiNum
reso = str_to_ipaddr(indiSr, indiNum)
; if indiSr = "192.168.1.10" -->indiNum= 0xc0a8010a
; if indiSr = "127.0.0.18" -->indiNum= 0x7f000012
```

Function str_val()

Function that returns, in numeric format, a value contained in a string.

Syntax	**real str_val (string strSrc, i32 nVal [, i32 base=10] [, i32 code=NULL])**
strSrc	String in which to search for the numeric value
nVal	Specifies which numeric datum, among those that the string contains, is to be converted (if

© 2025 Robox SpA

Syntax	**real str_val (string strSrc, i32 nVal [, i32 base=10] [, i32 code=NULL])**
	nVal=1 it will be the first numeric datum, if *nVal=2* we will refer to the second, ...)
base	Specifies the desired format for the converted value. (optional, default 10) Value / Format 10 / Decimal 16 / Hexadecimal
code	Returns a result code for the function: (optional, default NULL) Value / Description 0 / Operation performed successfully 1 / illegal*nVal* (<= 0) 2 / Data not found in *strSrc* 3 / Error during conversion
Result	Returns the specified numeric value (base 10) contained in the string
Validity	Rule, Task
Note	If the value specified by *nVal* does not match any number within the string, the returned value will be 0 and an error report will occur. The same behavior will occur if *nVal* < 1. If, during the search for the number to be converted, a "+" or "-" sign not followed by a number is found, *code* = 3 will be generated and the value returned will be 0

Examples

Example of use:

```
sr(4) = "axis 3 position = 3000"
rr(32) = str_val(sr(4), 1)
rr(200) = str_val(sr(4), 2)
rr(48) = str_val(sr(4), 3)
; After the execution of instructions rr(32)=3, rr(200)=3000, rr(48)
=0
```

Other examples:

Result	*nVal*	*base*	*code*	*strSrc*
16	1	10	0	0x10
16	1	16	0	0x10
10	1	10	0	0xA
10	1	16	0	0xA
0	1	16	2	Ah
0	1	16	2	hA
0	1	10	2	Ah
0	1	10	2	hA
123	1	17	0	123
123	1	5	0	123
291	1	16	0	123
2	1	10	0	2 athletes of 23 participants arrived. On average 8.69 %
23	2	10	0	2 athletes of 23 participants arrived. On average 8.69 %
8.69	3	10	0	2 athletes of 23 participants arrived. On average 8.69 %
0	4	10	2	2 athletes of 23 participants arrived. On average 8.69 %
3	1	10	0	3.45 ; incorrect comma
3.45	1	10	0	3.45
3	1	16	0	3.45 ; incorrect base
3	1	16	0	3.45 ; incorrect comma and base
0	1	10	3	+
0	1	10	3	-
0	1	16	0	+

© 2025 Robox SpA

Result	*nVal*	*base*	*code*	*strSrc*
0	1	16	0	-

Function sum8_mem()

Function that returns the sum of *n* elements of an array.

Syntax	**i32 sum8_mem (u8 buff, i32 nEl)**
buff	Array of u8
nEl	Number of elements in the array that you want to sum from the first position
Result	Returns the sum of the top *nEl* elements of the array
Validity	Rule, Task
Note	

Examples

Example of use:

```
int reso
u8 array[5]
; array = [1, 2, 3, 4, 5]

reso = sum8_mem 611(array, 0)   ; reso = 0
reso = sum8_mem(array, 1)   ; reso = 1
reso = sum8_mem(array, 2)   ; reso = 3
reso = sum8_mem(array, 3)   ; reso = 6
reso = sum8_mem(array, 4)   ; reso = 10
reso = sum8_mem(array, 5)   ; reso = 15
reso = sum8_mem(array, 12)  ; reso = 15
```

Function sum16_mem()

Function that returns the sum of *n* elements of an array.

Syntax	**i32 sum16_mem (u16 buff, i32 nEl)**
buff	Array of u16
nEl	Number of elements in the array that you want to sum from the first position
Result	Returns the sum of the top *nEl* elements of the array

© 2025 Robox SpA

Syntax	**i32 sum16_mem (u16 buff, i32 nEl)**
Validity	Rule, Task
Note	

Function sum32_mem()

Function that returns the sum of *n* elements of an array.

Syntax	**i32 sum32_mem (u32 buff, i32 nEl)**
buff	u32 array
nEl	Number of elements in the array that you want to sum from the first position
Result	Returns the sum of the top *nEl* elements of the array
Validity	Rule, Task
Note	

Function sumr_mem()

Function that returns the sum of *n* elements of an array.

Syntax	**i32 sumr_mem (real buff, i32 nEl)**
buff	Array of real
nEl	Number of array elements you want to sum from the first position
Result	Returns the sum of the top *nEl* elements of the array
Validity	Rule, Task
Note	

Function sw0()

This function is used to read or write (instruction) the first word (starting with the least significant) of the argument with sign extension.

Reading:

Syntax	**i16 sw0 (i16|i32|u16|u32|real arg)**
arg	Word argument. From RTE 33.7.1 it can be any expression

© 2025 Robox SpA

Syntax	**i16 sw0 (i16|i32|u16|u32|real arg)**
Result	Returns the first word of the argument with the sign
Validity	Rule, Task
Note	From RTE 33.9.0 accessing a non-existing word returns an error

Writing (instruction):

Syntax	**sw0 (i16|i32|u16|u32|real arg) = i16 value**
arg	Word argument
value	Value to be imposed
Validity	Rule, Task
Note	

Examples

Example of use:

```
I32 pluto, ris1, ris2
pluto = 0x12345678
; if pluto is 0x12345678
r(1) = sw0(pluto) ; r(1)= 0x5678
r(1) = sw1(pluto) ; r(1)= 0x1234
; if pluto is 0x80121234
r(1) = sw0(pluto) ; r(1)= 0x00001234
r(1) = sw1(pluto) ; r(1)= 0xffff8012

; with RTE 33.9.0:
I32 pippo = 123
sw4(pippo) = 9  ; error!
```

Function sw1()

This function is used to read or write (instruction) the second word (starting with the least significant) of the argument with sign extension.

Reading:

Syntax	**i16 sw1 (i32|u32|real arg)**
arg	Word argument. From RTE 33.7.1 it can be any expression
Result	Returns the second word of the argument with the sign

© 2025 Robox SpA

Syntax	**i16 sw1 (i32|u32|real arg)**
Validity	Rule, Task
Note	From RTE 33.9.0 accessing a non-existing word returns an error

Writing (instruction):

Syntax	**sw1 (i32|u32|real arg) = i16 value**
arg	Word argument
value	Value to be imposed
Validity	Rule, Task
Note	

Function sw2()

This function is used to read or write (instruction) the third word (starting with the least significant) of the argument with sign extension.

Reading:

Syntax	**i16 sw2 (real arg)**
arg	Argument word. From RTE 33.7.1 it can be any expression
Result	Returns the third word of the argument with the addition of the sign
Validity	Rule, Task
Note	From RTE 33.9.0 accessing a non-existing word returns an error

Writing (instruction):

Syntax	**sw2 (real arg) = i16 value**
arg	Word argument
value	Value to be imposed
Validity	Rule, Task
Note	

Function sw3()

This function is used to read or write (instruction) the fourth word (starting with the least significant) of the argument with sign extension.

Reading:

© 2025 Robox SpA

Syntax	**i16 sw3 (real arg)**
arg	Argument word. From RTE 33.7.1 it can be any expression
Result	Returns the fourth word of the argument with the addition of the sign
Validity	Rule, Task
Note	From RTE 33.9.0 accessing a non-existing word returns an error

Writing (instruction):

Syntax	**sw3 (real arg) = i16 value**
arg	Word argument
value	Value to be imposed
Validity	Rule, Task
Note	

Function tan()

Function that performs the tangent of a specified value.

Syntax	**real tan (real value)**
value	Tangent argument. **NOTE**: *value* must be in radians.
Result	Makes the tangent of *value*
Validity	Rule, Task
Note	If the argument is equal to K_PIMEZZI 17 or -K_PIMEZZI 17, the result will be ± 1E308 and an error report will not be issued

Examples

Example usage:

```
rr(5) = 0
rr(10) = tan 615(rr(5)) ; = 0
rr(10) = tan(K_PI/4 17) ; = 1
```

© 2025 Robox SpA

Function time_micro()

Function that returns the value of the time, in microseconds, elapsed since the Robox motion control was turned on.

Syntax	**real time_micro ()**
Result	Returns the elapsed time [us]
Validity	Rule, Task
Note	Can be used for duration measurements of portions of software, etc...

Examples

Example of use:

```
real calcTime

rr(100) = time_micro616()
calcTime = time_micro() - rr(100)
```

Function tm_et()

This function is used to read or set (Instruction) the current value (Elapsed Time) of the timer.

Reading:

WARNING: The value is readable only after the timer is started (e.g., after calling the tm_int751() function for the first time), otherwise alarm 9901 is generated.

Syntax	**real tm_et (timer name)**
name	Name of the timer of type timer900
Result	Returns the value, in milliseconds, of the current timer time
Validity	Rule, Task
Note	All timers, except the integral timer (tm_int751 ()), reset their current value the first time they are called

Writing (Instruction):

WARNING: The value can only be set before the timer is started (e.g., before calling the tm_int751() function for the first time), otherwise alarm 9901 is generated.

Syntax	**tm_et (timer name, real valueEt)**
name	Name of the timer of type timer900
valueEt	Value of the time, in milliseconds, to be set
Validity	Rule, Task

© 2025 Robox SpA

Syntax	**tm_et (timer name, real valueEt)**
Note	

Examples

Example of use:

```
; Task that uses the timer tm_int to measure how long the control
remains on.
; The elapsed time is continuously written into an nvrr(1), so that
the value is not lost in the case of a power down.
; Each time it is turned back on, the timer reloads the value in the
nvrr and starts counting again from that value.

$task 1

timer 900 timer1
tm_et 616 (timer1, nvrr(1))      ; nvrr(1) -> timer1.et
tm_int 751 (timer1, 1)           ; activate the integral timer

__MAIN_LOOP__
   nvrr(1) = tm_et(timer1) ; timer1.et -> nvrr(1)
END_MAIN_LOOP
```

Function tm_pt()

This function is used to read or set (instruction) the preset value of the timer.

Reading:

Syntax	**real tm_pt (timer name)**
name	Name of the timer of type timer 900
Result	Returns the value, in milliseconds, of timer preset
Validity	Rule, Task
Note	

Writing (Instruction):

Syntax	**tm_pt (timer name, real presetTm)**
name	Name of the timer of type timer 900
presetTm	Value of the preset, in milliseconds, to be set
Validity	Rule, Task
Note	

Examples

Example usage:

```
tm_pt(T1, rr(1000)) ; I write the preset value contained in the
real register rr(1000) into timer T1
tm_pt(tm[idx], 1000) ; I write the 1000 preset value into the idx
[array of timers] timer
valuePreset = tm_tp(T1) ; I write the preset value into the variable
valuePreset
```

Function tm_pt2()

This function is used to read or set (Instruction) the value of the second square timer preset.

Reading:

Syntax	**real tm_pt (timer name)**
name	Name of the timer of type timer
Result	Returns the value, in milliseconds, of timer preset
Validity	Rule, Task
Note	

Write (Instruction):

Syntax	**tm_pt2 (timer name, real presetTm)**
name	Name of the timer of type timer
presetTm	Value of the preset, in milliseconds, to be set
Validity	Rule, Task
Note	

Examples

Example usage:

```
tm_pt2(T1, rr(1000)) ; I write the preset value contained in the
real register rr(1000) into timer T1
valuePreset2 = tm_tp2(T1) ; I write the value of preset 2 into the
variable valuePreset2
```

© 2025 Robox SpA

Function tm_q()

Function to read the status value of the timer output.

Syntax	**bool tm_q (timer name)**
name	Name of the timer of type timer 900
Result	Returns the status value of the timer output
Validity	Rule, Task
Note	

Function tobcd()

Function that converts an integer numeric value, expressed in binary format, to Binary-Coded Decimal (BCD) format.

Syntax	**i32 tobcd (i32 value)**
value	Variable representing the character (0-:-255) or EOF (-1)
Result	Returns the converted value in BCD format
Validity	Rule, Task
Note	Since the output value is encoded in 4 bytes, the field of variability of the output value is: 0-:-99999999 always positive

The function operates as follows:

- The input real-type expression is converted to an integer quantity
- The absolute value of the integer quantity is executed
- If the value is greater than 99999999, it is restricted to 99999999
- The resulting value is converted to BCD

Diagnostics

- Generates 9900 ill. Arg A:1 T:x ST:xxx if illegal argument (argument > 99999999)

Examples

Example of use:

```
; Assuming we want to write the number 1234 encoded in BCD on
output_word port 1, we will write:
out_w(1) = tobcd 619(1234)
```

© 2025 Robox SpA

Function tobin()

Function that converts an integer numeric value, expressed in Binary-Coded Decimal (BCD) format, to the binary format.

Syntax	**i32 tobin (i32 value)**
value	BCD value that you want to convert
result	Returns the converted value in binary format
Validity	Rule, Task
Note	Binary format is used by operators 774 in the R3 language

Diagnostics

- Generates 9900 ill. Arg A:1 T:x ST:xxx if illegal argument (one of the digits is > 9)

Examples

Example usage:

```
r(33) = tobin 620 (inp_w(1))
; 16bits of input_word port 1 are interpreted as 4 BCD-encoded
numbers converted to pure binary and written to r(33)
; In the example below, in which it was imagined that we wanted to
read from input port input_word 1 a unit,
; encoded BCD whose range is 0-999 (12 bits) and also imagine that
16th bit represents the valid data code(DATA_VALID)
r(33) = inp_w(1)
; downstream of this instruction the 16 bits of input port 1 are
written to the integer register r(33)
if(r(33) r_and 777 0X8000)
 ; check DATA_VALID: if DATA_VALID present read code and convert
 r(34) = r(33) r_and 777 0X0FFFH
 ; extract the 12 least significant bits
  r(35) = tobin(r(34))
 ; convert to pure binary
endif
; after instruction execution, assuming you have read the value
0X8634H, the registers will contain:
```

Register	**R(33)**	**R(34)**	**R(35)**
Hexadecimal	8634	634	27A
Decimal	-31180	1588	634

© 2025 Robox SpA

Function todeg()

Function that converts a numerical value from radians to degrees.

Syntax	**real todeg (real value)**
value	Represents the value, expressed in radians, that you want to convert
result	Returns the converted value in degrees
Validity	Rule, Task
Note	All trigonometric functions operate on arguments expressed in radians

Examples

Example usage:

```
angle = todeg(K_PI / 6)
; after the instruction is executed, the content of the local
variable angle will be 30
```

Function tolower()

Function that converts a character to lower case.

Syntax	**i32 tolower (i32 ch)**
ch	Character to convert
Result	Returns the converted character
Validity	Rule, Task
Note	

Function torad()

Function that converts a numerical value from degrees to radians.

Syntax	**real torad (real value)**
value	Represents the value, expressed in degrees, that you want to convert
result	Returns the converted value in radians
Validity	Rule, Task
Note	All trigonometric functions operate on arguments expressed in radians

Examples

© 2025 Robox SpA

Example usage:

```
sinTeta = sin(torad(30))
; after the instruction is executed, the contents of the local
variable sinTeta will be 0.5
```

Function toupper()

Function that converts a character to uppercase.

Syntax	**i32 toupper (i32 ch)**
ch	Character to convert
Result	Returns the converted character
Validity	Rule, Task
Note	

Function tracking()

Function to bring the controlled variable (typically an axis) to the specified target with the specified velocity.

The function, runtime, checks that the maximum speed and acceleration specified for the controlled variable are not exceeded.

The function also checks, again runtime, that the controlled variable does not go outside the specified end of strokes by performing a controlled stop on them.

To override the control of strokes set them to the same value.

Syntax	**i32 tracking (stru_track stru)**
stru	Structure of type stru_track for data exchange
Result	Returns:

Value	Meaning
-9	Parameter *delta* <= 0
-6 [M_IN_ALARM]	*acc* parameter null
-5 [M_ILLEGAL_ARGS]	*maxspeed* parameter null
5 [M_FOLLOWING]	Target not reached - tracking in progress
6 [M_REACHED]	Target reached - one of the two position limits reached

© 2025 Robox SpA

Syntax	**i32 tracking (stru_track stru)**
	<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>8 [M_SYNCHRONIZED]</td><td>Target reached - moving in torque</td></tr><tr><td>9 [M_ACCEL_LIMITED]</td><td>Target not reached - tracking in progress - acceleration limitation</td></tr><tr><td>10 [M_SPEED_LIMITED]</td><td>Target not reached - tracking in progress - speed limitation</td></tr><tr><td>17 [M_DECEL]</td><td>Target not reached - deceleration to reach either position limit</td></tr><tr><td>19 [M_OUT_OF_LIMITS]</td><td>Target not reached - attempt to move further out of position limits</td></tr></table>
Validity	Rule
Note	If you want to disable limits control, impose them the same value

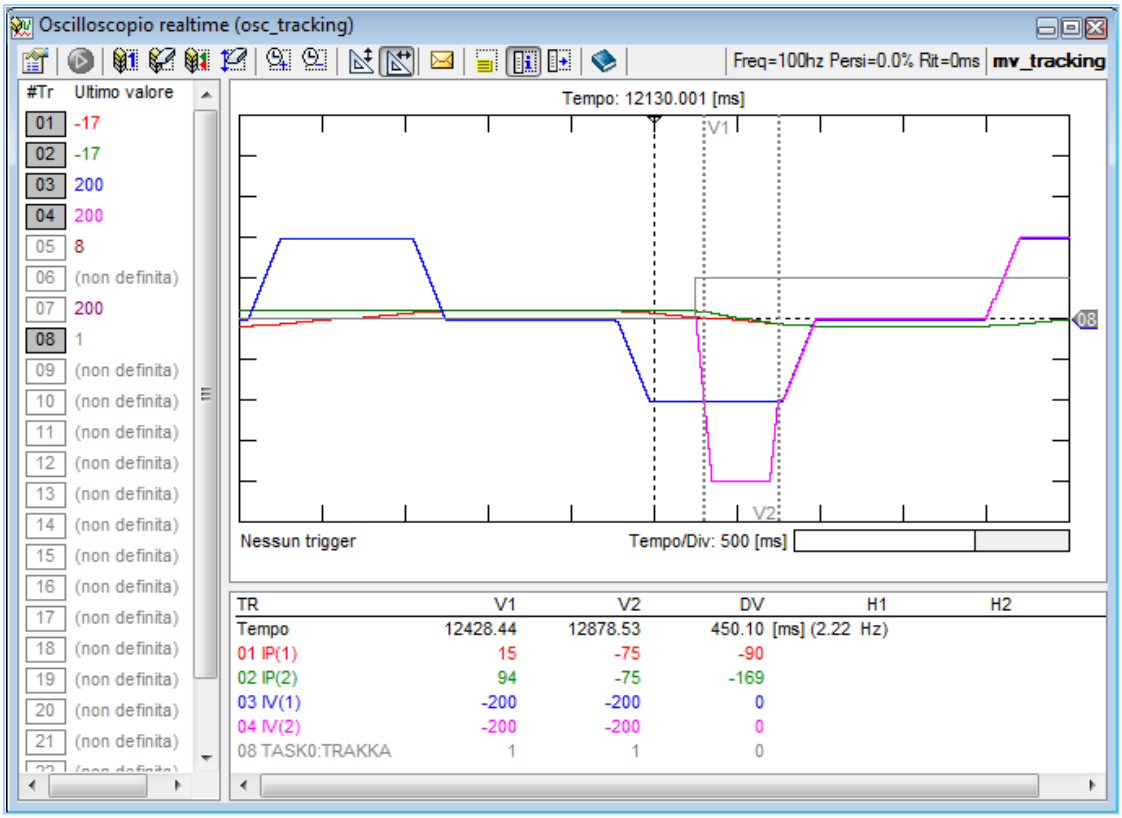
Examples

Example of use:

```
RULE R_AUTO_2
axes(AX_2)
motion
    if(first_time 321())
    endif
    if(trakka)
        ; Let's track AXES 1 considering  AXES 2 limitations
        trk.delta = SI
        trk.desspeed = IV(AX_1)
        trk.despos = IP(AX_1)
        trk.actpos = IP(AX_2)
        trk.actspeed = IV(AX_2)
        trk.maxspeed = MAX_SPE(AX_2)
        trk.acc = MAX_ACC(AX_2)
        trk.posmin = MIN_STR(AX_2)
        trk.posmax = MAX_STR(AX_2)
        ; call the function
        reso = tracking 622(trk)
        ; link to the axe
        IP(AX_2) = trk.actpos
    else
        ; stop the axe
```

© 2025 Robox SpA

```
        IV(AX_2) = ramp(IV(AX_2), 0, MAX_ACC(AX_2))
    endif
end_motion
END_RULE
```



Function trunc()

Function that returns the integer part of its argument.

Syntax	**i32 trunc (real value)**
value	Value or expression from which to get the integer part
result	Returns the integer part of the argument
Validity	Rule, Task
Note	Always returns a value approximating the integer less than or equal to the input value (e.g. 5.3 -> 5, 5.8 -> 5, -5.3 -> -6, ..)

Examples

Example of use:

© 2025 Robox SpA

```
r(100) = trunc(3.141598)     ; r(100) = 3
r(101) = trunc(3.141598 * 2 / 6.0 * sin(torad(90)))     ; r(101)
= 1
```

Function udp_close()

Function for closing an UDP communication.

Syntax	**i32 udp_close (i32 idx)**
idx	Number of the handler of the UDP communication to be closed
Result	Returns:

Value	Meaning
0	Operation performed successfully
-1	Illegal *idx* value
-2	Socket already closed
-1000 -:- -2000	Error code of the S.O. close() function, with an offset of 1000

To see more specifically some values indicating error, see Returned Errors Table

Validity	Rule, Task
Note	

For a concrete example see in RDE environment: Workspace -> Specials -> Examples Library.

Returned errors table

Value	Meaning
-1004	[EINTR] Function interrupted by a system call
-1009	[EBADF] Incorrect file number

NOTE: For the complete table of error codes see Function error codes paying attention to the offset.

Function udp_open_client()

Function for opening a client-type UDP communication.

The client sends messages to a specified destination station and eventually reads the response from the same station.

Syntax	**i32 udp_open_client (u32 nPort, u32 nAddr)**
nPort	Server port number (remote) to which the client will send its messages
nAddr	IP address (remote) to which the client will send its messages
Result	Returns:

Value	Meaning
0-:-99	Operation performed successfully. The communication ID is returned
-1	Illegal *nPort*
-2	*nAddr* illegal. Must be different from 0
-3	All task connection table entries are already busy, impossible to establish a new connection (max 10 connections)
-1000 -:- -2000	Socket opening failed: returns the code provided by the S.O. socket() function, with offset 1000
-2000 -:- -3000	S.O. function connect() failed: returns the code provided by connect(), with offset 2000

To see some returned errors more specifically, see Returned errors table 627

Validity	Rule, Task
Note	If the result is negative, it should not be used in the other UDP communication management functions

For a concrete example see in the RDE environment: Workspace -> Specials -> Examples Library.

© 2025 Robox SpA

Returned errors table

Value	Meaning
-1013	[EACCES] Denied permission to create socket of specified type and protocol
-1023	[ENFILE] System file table is full
-1024	[EMFILE] Pre-process description table is full
-1091	[EPROTOTYPE] Socket type is not supported by the protocol
-1093	[EPROTONOSUPPORT] The protocol family is not supported or the protocol itself is not supported
-1097	[EAFNOSUPPORT] The domain is not supported
-1105	[ENOBUFS] Insufficient buffer space
-2002	[ENOENT] The socket does not exist
-2004	[EINTR] The connection attempt was aborted
-2009	[EBADF] Id invalid
-2013	[EACCES] An attempt was made to make a connection to a broadcast address without the socket having been enabled for broadcast
-2014	[EFAULT] Has an invalid address
-2022	[EINVAL] Socket address out of the allowed range
-2040	[ELOOP] Too many symbolic links encountered during path translation
-2088	[ENOTSOCK] Invalid id, does not refer to a socket
-2097	[EAFNOSUPPORT] The address does not have a valid address family correct in its field
-2098	[EADDRINUSE] Another socket is already listening on the same port
-2101	[ENETUNREACH] The network is unreachable
-2106	[EISCONN] The socket is already connected

© 2025 Robox SpA

Value	Meaning
-2110	[ETIMEDOUT] A timeout occurred during the connection attempt. On request for SYN (n attempts every x seconds) there was no response (server busy)
-2111	[ECONNREFUSED] There is no one listening on the remote address (on request from SYN the client received an RST sent directly from the server's TCP management when something is wrong) or the server has reached the maximum number of possible connections
-2114	[EALREADY] The socket is not blocking and a previous connection attempt has not yet concluded
-2115	[EINPROGRESS] The socket is not blocking and the connection cannot be concluded immediately

NOTE: For the complete table of error codes see Error codes functions 57 paying attention at the offset.

Function udp_open_server()

Function for opening a server-type UDP communication.

The server remains listening for messages from several stations and responds to each of them if necessary. Listening occurs on the local port specified by *nPort* and can be limited to the local interface specified by *nAddr* or extended to all local interfaces.

Syntax	**i32 udp_open_server (u32 nPort [, i32 nAddr=ALL])**
nPort	Number of the (local) port on which the server is listening
nAddr	IP (local) address of the interface on which the server is listening. (optional, default to all local interfaces)
Result	Returns:

Value	Meaning
0-:-99	Operation performed successfully. The communication ID is returned
-1	Illegal *nPort*
-3	All task connection table entries are already committed,

Syntax	**i32 udp_open_server (u32 nPort [, i32 nAddr=ALL])**
	<table><tr><th>Value</th><th>Meaning</th></tr><tr><td></td><td>impossible to establish a new connection (max 10 connections)</td></tr><tr><td>-1000 -:- -2000</td><td>Socket opening failed: returns the code provided by the S.O. socket() function, with offset 1000</td></tr><tr><td>-2000 -:- -3000</td><td>S.O. function bind() failed: returns the code provided by bind(), with offset 2000</td></tr></table> To see some returned errors more specifically, see Returned errors table 629
Validity	Rule, Task
Note	If the result is negative, it should not be used in the other UDP communication handling functions

For a concrete example see in RDE environment: Workspace -> Specials -> Examples Library.

Returned errors table

Value	Meaning
-1013	[EACCES] Permission to create socket of the specified type and protocol denied
-1023	[ENFILE] System file table is full
-1024	[EMFILE] Pre-process description table is full
-1091	[EPROTOTYPE] The socket type is not supported by the protocol
-1093	[EPROTONOSUPPORT] The protocol family is not supported or the protocol itself is not supported
-1097	[EAFNOSUPPORT] The domain is not supported
-1105	[ENOBUFS] Insufficient buffer space
-2002	[ENOENT] The socket does not exist

© 2025 Robox SpA

Value	Meaning
-2004	[EINTR] The connection attempt was aborted
-2009	[EBADF] Id invalid
-2013	[EACCES] An attempt was made to make a connection to a broadcast address without the socket having been enabled for broadcast
-2014	[EFAULT] Has an invalid address
-2022	[EINVAL] Socket address out of the allowed range
-2040	[ELOOP] Too many symbolic links encountered during path translation
-2088	[ENOTSOCK] Invalid id, not referring to a socket
-2097	[EAFNOSUPPORT] Address does not have a valid address family corrected in its field
-2098	[EADDRINUSE] Another socket is already listening on the same port
-2101	[ENETUNREACH] The network is not reachable
-2106	[EISCONN] The socket is already connected
-2110	[ETIMEDOUT] Timeout occurred during connection attempt. On request for SYN (n attempts every x seconds) there was no response (server busy)
-2111	[ECONNREFUSED] There is no one listening on the remote address (on request from SYN the client received an RST sent directly from the server's TCP management when something is wrong) or the server has reached the maximum number of possible connections
-2114	[EALREADY] The socket is not blocking and a previous connection attempt has not yet concluded
-2115	[EINPROGRESS] The socket is not blocking and the connection cannot be concluded immediately

NOTE: For the complete table of error codes see Error codes functions 57 taking offset into account.

Function udp_recv()

Function for receiving an UDP message from a server.

The station (nPort and nAddr) was previously specified via an udp_open_client 626().

Syntax	**i32 udp_recv (i32 idx, buff, i32 buffLen)**
idx	Number of the UDP communication handler
buff	Generic buffer intended to contain the received data packet. It can be a string variable, struct_p 889 (packed structure), or an array of struct_p. In the case of struct_p, the variable must be declared global
buffLen	Maximum number of bytes to receive
Result	Returns: Value / Meaning >= 0 / Operation performed successfully. The number of characters received is returned -1 / Illegal *idx* -2 / Socket not open -3 / Socket open as server and not as client -1000 -:- -2000 / Error code of the S.O. function recv(), with offset 1000 To see some returned errors more specifically, see Returned Errors Table 631
Validity	Rule, Task
Note	The udp_recv() can only be used for client communications

For a concrete example see in RDE environment: Workspace -> Specials -> Examples Library.

Returned errors table

Value	Meaning
-1004	[EINTR] Reading was interrupted before any data was received
-1009	[EBADF] Id invalid

Value	Meaning
-1011	[EAGAIN] Retry. Socket was non-blocking and reception would block execution, or a timeout on reception was imposed and that timeout expired before any data was received
-1014	[EFAULT] Buffer pointer points to a memory area outside the process space
-1022	[EINVAL] The total I/O length is greater than the maximum that can be represented by the returned type
-1088	[ENOTSOCK] Invalid id, does not refer to a socket
-1105	[ENOBUFS] The message was not sent because it would have exceeded the maximum buffer size
-1107	[ENOTCONN] The connection has not yet been created
-1122	[EMSGSIZE] Message too long

NOTE: For the complete table of error codes see Error codes functions 57 paying attention at the offset.

Function udp_recv_from()

Function to receive an UDP message with the possibility to know the port and IP address of the transmitter.

The function can be used for both client and server communications. Specifically, the server uses the *rPort* and *rIp* information returned by this function to send an eventual response to the client station via the udp_send_to 640().

Syntax	**i32 udp_recv_from (i32 idx, buff, i32 buffLen, u32 rPort, u32 rIp)**
idx	Number of the handler of the UDP communication
buff	Generic buffer intended to contain the received data packet. It can be a string variable, a struct_p 889 (packed structure), or an array of struct_p. In the case of struct_p, the variable must be declared global
buffLen	Maximum number of bytes to receive
rPort	Number of the remote port from which data was received (its contents are written by this function)

© 2025 Robox SpA

Syntax	**i32 udp_recv_from (i32 idx, buff, i32 buffLen, u32 rPort, u32 rIp)**
rIp	Remote IP address from which the data was received (its contents are written by this function; it is not necessary to specify it before the call)
Result	Returns: Value / Meaning >= 0 / Operation performed successfully. The number of characters received is returned -1 / Illegal *idx* -2 / Socket not open -1000 -:- -2000 / Error code of the S.O. function recv_from(), with offset 1000 To see some returned errors more specifically, see Returned Errors Table 633
Validity	Rule, Task
Note	A place must always be reserved at the head of the receive data buffer for a u32, which contains the identifier *msgId*. This identifier must be retransmitted without modification (via the udp_send_to 640() function) at the head of the buffer containing the response. Without this identifier, a client using the udp_send_notify 638() and udp_recv_notify 634() functions will not be able to identify the response as correct

For a concrete example see in the RDE environment: Workspace -> Specials -> Examples Library.

Returned errors table

Value	Meaning
-1004	[EINTR] Reading was interrupted before any data was received
-1009	[EBADF] Id invalid
-1011	[EAGAIN] Retry. Socket was non-blocking and reception would block execution, or a timeout on reception was imposed and that timeout expired before any data was received

© 2025 Robox SpA

Value	Meaning
-1014	[EFAULT] Buffer pointer points to a memory area outside the process space
-1022	[EINVAL] The total I/O length is greater than the maximum that can be represented by the returned type
-1088	[ENOTSOCK] Invalid id, does not refer to a socket
-1105	[ENOBUFS] The message was not sent because it would have exceeded the maximum buffer size
-1107	[ENOTCONN] The connection has not yet been created
-1122	[EMSGSIZE] Message too long

NOTE: For the complete table of error codes see Error codes functions 57 paying attention at the offset.

Function udp_recv_notify()

Function for receiving a response from a server to a previously sent UDP message.

This call makes sense only after calling the corresponding udp_send_notify 638() function. The function can only be used for communication from the client.

Syntax	**i32 udp_recv_notify(i32 idx, buff, i32 buffLen, real tOutRx, real tOutTot)**
idx	Number of the handler of the UDP communication
buff	Generic buffer intended to contain the received data packet. It can be a string variable, a struct_p 889 (packed structure), or an array of struct_p. In the case of struct_p, the variable must be declared global. If struct, the first member must be declared of type u32 and represents the message identifier *msgId*; the value is assigned by the udp_send_notify 638() function. *msgId* must be used server-side in the response message and is used to ensure routing to the correct client. All other data is for user use. The maximum size of this structure is: MAX_UDP_DATA=508 bytes
buffLen	Maximum number of bytes to receive (including *msgId*)
tOutRx	When this timeout time expires, the message is automatically retransmitted

Syntax	**i32 udp_recv_notify(i32 idx, buff, i32 buffLen, real tOutRx, real tOutTot)**
tOutTot	When this timeout time expires, the function is aborted. You can call the function with *tOutTot=0* to cause the function to be aborted
Result	Returns: **Value** / **Meaning** > 0 — Operation performed successfully. The number of characters sent is returned 0 — Waiting for response -1 — Illegal *idx* -2 — Socket not open -3 — Socket open as server and not as client -4 — *buffLen* greater than the maximum allowed -5 — No udp_send_notify() results for the specified *idx* (638) -6 — *tOutTot* expired -1000 -:- -2000 — Error code of S.O. function sendto(), with offset 1000 (returned error in S.O. function sendto() called to retry transmission after tmOutRx) To see some returned errors more specifically, see Returned Error Table (635)
Validity	Rule, Task
Note	The two timeouts will only be able to work correctly if the function is called at a sufficiently high frequency relative to the specified times

For a concrete example see in the RDE environment: Workspace -> Specials -> Examples Library.

Returned errors table

Value	Meaning
-1004	[EINTR] Reading was interrupted before any data was received
-1009	[EBADF] Id invalid
-1011	[EAGAIN] Retry. Socket was non-blocking and reception would block execution, or a timeout on reception was imposed and that timeout expired before any data was received
-1014	[EFAULT] Buffer pointer points to a memory area outside the process space
-1022	[EINVAL] The total I/O length is greater than the maximum that can be represented by the returned type
-1088	[ENOTSOCK] Invalid id, does not refer to a socket
-1105	[ENOBUFS] The message was not sent because it would have exceeded the maximum buffer size
-1107	[ENOTCONN] The connection has not yet been created
-1122	[EMSGSIZE] Message too long

NOTE: For the complete table of error codes see Error codes functions 57 paying attention at the offset.

Function udp_send()

Function for sending an UDP message to a server.

The station (nPort and nAddr) was previously specified via an udp_open_client 626().

Syntax	**i32 udp_send (i32 idx, buff, i32 dataLen)**
idx	Number of the handler of the UDP communication
buff	Generic buffer containing the data packet to be transmitted. It can be a string variable, a struct_p 889 (packed structure), or an array of struct_p. In the case of struct_p, the variable must be declared global
DataLen	Number of total bytes to be transmitted
Result	Returns:

Syntax	**i32 udp_send (i32 idx, buff, i32 dataLen)**
	<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>>= 0</td><td>Operation performed successfully. The number of characters sent is returned. The message is always sent in full, so the number of bytes sent is always equal to *dataLen* unless there are errors</td></tr><tr><td>-1</td><td>Illegal *idx*</td></tr><tr><td>-2</td><td>Socket not open</td></tr><tr><td>-3</td><td>Socket open as server and not as client</td></tr><tr><td>-1000 -:- -2000</td><td>Error code of the S.O. function sendto(), with offset 1000</td></tr></table> To see some returned errors more specifically, see Returned Error Table 637
Validity	Rule, Task
Note	The udp_send() can only be used for communication from the client

For a concrete example see in RDE environment: Workspace -> Specials -> Examples Library.

Returned errors table

Value	Meaning
-1009	[EBADF] Id invalid
-1011	[EAGAIN or EWOULDBLOCK] Retry. The socket was non-blocking and receiving would block execution, or a timeout on receiving was imposed and that timeout expired before any data was received
-1013	[EACCES] The SO_BROADCAST option was not imposed on the socket and an address set is broadcast type
-1014	[EFAULT] Buffer pointer points to a memory area outside the process space
-1022	[EINVAL] The total I/O length is greater than the maximum that can be represented

© 2025 Robox SpA

Value	Meaning
	by the returned type
-1032	[EPIPE] The socket has lost its connection
-1088	[ENOTSOCK] Invalid id, does not refer to a socket
-1089	[EDESTADDRREQ] In an unconnected socket, the address was not specified
-1097	[EAFNOSUPPORT] The address specified in the address family cannot be used by this socket
-1105	[ENOBUFS] The message was not sent because it would have exceeded the maximum buffer size
-1106	[EISCONN] A destination address was specified and the socket is already connected
-1112	[EHOSTDOWN] The destination is a host on the local subnet and is not responding to the ARP protocol
-1113	[EHOSTUNREACH] Destination not reachable
-1122	[EMSGSIZE] Message too long

NOTE: For the complete table of error codes see Function error codes 57 paying attention at the offset.

Function udp_send_notify()

Function for sending a UDP message with notification to a previously specified station.

The message mandatorily includes a response, which must be read via the udp_recv_notify 634() function. The function can be used only for communication from the client.

Syntax	**i32 udp_send_notify (i32 idx, buff, i32 buffLen)**
idx	UDP communication handler number
buff	Generic buffer containing the data packet to be transmitted. It can be a string variable, a struct_p 889 (packed structure), or an array of struct_p. In the case of struct_p, the variable must be declared global. In the struct, the first member must be declared of type u32 and represents the message identifier *msgId*; the value is assigned by the udp_send_notify 638() function. *msgId*

Syntax	**i32 udp_send_notify (i32 idx, buff, i32 buffLen)**
	must be used server-side in the response message and is used to ensure routing to the correct client. All other data is for user use. The maximum size of this structure is: MAX_UDP_DATA=508 bytes
buffLen	Number of total bytes to be transmitted (including *msgId*)
Result	Returns: Value / Meaning >= 0 / Operation performed successfully. The number of characters sent is returned -1 / Illegal *idx* -2 / Socket not open -3 / Socket open as server and not as client -4 / *buffLen* greater than maximum allowed -5 / An udp_send_notify 638 () already in progress for the specified *idx* -1000 -:- -2000 / Error code of the S.O. function sendto(), with offset 1000 To see some returned errors more specifically, see Returned Error Table 639
Validity	Rule, Task
Note	Only one udp_send_notify() function can be active simultaneously toward each server (i.e., for each open *idx*)

For a concrete example see in the RDE environment: Workspace -> Specials -> Examples Library.

Returned errors table

Value	Meaning
-1009	[EBADF] Id invalid
-1011	[EAGAIN or EWOULDBLOCK] Retry. The socket was non-blocking and receiving

© 2025 Robox SpA

Value	Meaning
	would block execution, or a timeout on receiving was imposed and that timeout expired before any data was received
-1013	[EACCES] The SO_BROADCAST option was not imposed on the socket and an address set is broadcast type
-1014	[EFAULT] The buffer pointer points to an area of memory outside the process space
-1022	[EINVAL] The total I/O length is greater than the maximum that can be represented by the returned type
-1032	[EPIPE] The socket has lost connection
-1088	[ENOTSOCK] Invalid id, does not refer to a socket
-1089	[EDESTADDRREQ] In an unconnected socket, the address was not specified
-1097	[EAFNOSUPPORT] The address specified in the address family cannot be used by this socket
-1105	[ENOBUFS] The message was not sent because it would have exceeded the maximum buffer size
-1106	[EISCONN] A destination address was specified and the socket is already connected
-1112	[EHOSTDOWN] The destination is a host on the local subnet and does not respond to the ARP protocol
-1113	[EHOSTUNREACH] Destination not reachable
-1122	[EMSGSIZE] Message too long

NOTE: For the complete table of error codes see Error codes functions 57 paying attention at the offset.

Function udp_send_to()

Function for sending an UDP message to a specified station (nPort and nAddr).

Syntax	**i32 udp_send_to (i32 idx, buff, i32 dataLen, u32 rPort, u32 rIp)**
idx	Number of the handler of the UDP communication

© 2025 Robox SpA

Syntax	**i32 udp_send_to (i32 idx, buff, i32 dataLen, u32 rPort, u32 rIp)**
buff	Generic buffer containing the data packet to be transmitted. It can be a string variable, a struct_p 889 (packed structure), or an array of struct_p. In the case of struct_p, the variable must be declared global
dataLen	Number of total bytes to be transmitted
rPort	Number of the remote port to which the data is to be transmitted
rIp	IP address to which to transmit the data
Result	Returns:

Value	Meaning
>= 0	Operation performed successfully. The number of characters sent is returned. The message is always sent in full, so the number of bytes sent is always equal to *dataLen* unless there are errors
-1	Invalid port number
-2	Invalid address. Must be different from 0
-3	Illegal *idx*
-4	Socket not open
-1000 -:- -2000	Error code of the S.O. function sendto(), with offset 1000

To see some returned errors more specifically, see Returned Error Table 641

Validity	Rule, Task
Note	The function can be used for client and server communications

For a concrete example see in RDE environment: Workspace -> Specials -> Examples Library.

Returned errors table

Value	Meaning
-1009	[EBADF] Id invalid
-1011	[EAGAIN or EWOULDBLOCK] Retry. The socket was non-blocking and receiving would block execution, or a timeout on receiving was imposed and that timeout expired before any data was received
-1013	[EACCES] The SO_BROADCAST option was not imposed on the socket and an address set is broadcast type
-1014	[EFAULT] Buffer pointer points to a memory area outside the process space
-1022	[EINVAL] The total I/O length is greater than the maximum that can be represented by the returned type
-1032	[EPIPE] The socket has lost its connection
-1088	[ENOTSOCK] Invalid id, does not refer to a socket
-1089	[EDESTADDRREQ] In an unconnected socket, the address was not specified
-1097	[EAFNOSUPPORT] The address specified in the address family cannot be used by this socket
-1105	[ENOBUFS] The message was not sent because it would have exceeded the maximum buffer size
-1106	[EISCONN] A destination address was specified and the socket is already connected
-1112	[EHOSTDOWN] The destination is a host on the local subnet and is not responding to the ARP protocol
-1113	[EHOSTUNREACH] Destination not reachable
-1122	[EMSGSIZE] Message too long

NOTE: For the complete table of error codes see Function error codes 57 paying attention at the offset.

Function virt_io()

Function to define a virtual input and output zone.

These are 16-bit entities not directly connected to any physical device but with the possibilities to have its own value forced.

Access is the same as for normal inp_w and out_w. It is the user's responsibility to set the actual state of the inputs with the appropriate instruction.

The virtual block will occupy all addresses between *ioBase* and(*ioBase* + maximum 364 (*nWordIn*, *nWordOut*) - 1).

Syntax	**i32 virt_io (i32 ioBase, i32 nWordIn, i32 nWordOut)**
ioBase	Minor index of the I/O group
nWordIn	Number of input words handled for this block
nWordOut	Number of output words handled for this block
Result	Returns:

Value	Meaning
0	Operation performed successfully
-1	Index *ioBase* less than 256
-2	Number of words <= 0
-3	I/O area already in use
-4	Memory full. Unable to create new virtual I/O
-5	Exhausted virtual I/O managers
-10	Function can only be used by tasks 1-:-8

Validity	Task
Note	Maximum 4 different blocks can be defined

Examples

Example of use:

```
; =======
; in task
; =======
int iobase = 2000
int n_win  = 90
int n_wout = 90
int esitoVIO
; initialization virtual I/Os
esitoVIO = virt_io 642(iobase, n_win, n_wout)
; initialization control
```

```
if(esitoVIO < 0)
    ; initialization failed
    ; ...
endif
; ==============
; in task or rule
; ==============
int desIWsts, owSts
int iw, ow
int esitoVIN, esitoVOUT
esitoVIN = set_virt_iw(2017, r(2345))
; result control
if(not esitoVIN)
    ; result failed
    ; ...
endif
esitoVOUT = get_virt_ow(2017)
```

Function w0()

This function is used to read or write (instruction) the first word (starting with the least significant) of the argument.

Reading:

Syntax	**u16 w0 (i16|i32|u16|u32|real arg)**
arg	Word argument. From RTE 33.7.1 it can be any expression
Result	Returns the first word of the argument
Validity	Rule, Task
Note	From RTE 33.9.0 accessing a non-existing word returns an error

Writing (Instruction):

Syntax	**w0 (i16|i32|u16|u32|real arg) = u16 value**
arg	Word argument
value	Value to be imposed
Validity	Rule, Task
Note	This instruction cannot be used to set the value of a register (e.g., w0(r(1))=val)

Examples

Example of use:

```
I32 pluto
; if pluto is 0x12345678
r(1) = w0 644 (pluto) ; r(1)= 0x5678
r(1) = w1 645 (pluto) ; r(1)= 0x1234
; defining a variable max as integer 32 bit
I32 max
w0 759 (max) = 0x1234
w1 759 (max) = 0x5678
; max will be = 0x56781234

; from RTE 33.9.0:
I32 pippo = 123
w4(pippo) = 9  ; error!
```

Function w1()

This function is used to read or write (instruction) the second word (starting with the least significant) of the argument.

Reading:

Syntax	**u16 w1 (i32|u32|real arg)**
arg	Word argument. From RTE 33.7.1 it can be any expression
Result	Returns the second word of the argument
Validity	Rule, Task
Note	From RTE 33.9.0 accessing a non-existing word returns an error

Writing (Instruction):

Syntax	**w1 (i32|u32|real arg) = u16 value**
arg	Word argument
value	Value to be imposed
Validity	Rule, Task
Note	This instruction cannot be used to set the value of a register (e.g., w1(r(1))=val)

Function w2()

This function is used to read or write (instruction) the third word (starting with the least significant) of the argument.

Reading:

© 2025 Robox SpA

Syntax	**u16 w2 (real arg)**
arg	Argument word. From RTE 33.7.1 it can be any expression
Result	Returns the third word of the argument
Validity	Rule, Task
Note	From RTE 33.9.0 accessing a non-existing word returns an error

Writing (Instruction):

Syntax	**w2 (real arg) = u16 value**
arg	Word argument
value	Value to be imposed
Validity	Rule, Task
Note	This instruction cannot be used to set the value of a register (e.g., w2(r(1))=val)

Function w3()

This function is used to read or write (instruction) the fourth word (starting with the least significant) of the argument.

Reading:

Syntax	**u16 w3 (real arg)**
arg	Argument word. From RTE 33.7.1 it can be any expression
Result	Returns the fourth word of the argument
Validity	Rule, Task
Note	From RTE 33.9.0 accessing a non-existing word returns an error

Writing (Instruction):

Syntax	**w3 (real arg) = u16 value**
arg	Word argument
value	Value to be imposed
Validity	Rule, Task
Note	This instruction cannot be used to set the value of a register (e.g., w3(r(1))=val)

© 2025 Robox SpA

Function wbe2le()

Function that converts big endian word (x16 or x32) to little endian word (x16).

Syntax	**i16 wbe2le (i16|i32 data)** **u16 wbe2le (u16|u32 data)**
data	Data input
Result	Return value
Validity	Rule, Task
Note	

Examples

Example of use:

```
U16 input_1
U32 input_2
U16 result
input_1 = 0x0102
input_2 = 0x01020304

result = wbe2le(input_1)
; result = 0x0201
result = wbe2le(input_2)
; result = 0x0403
```

Function wbit()

Function to set one or more bits in an integer-type data if the condition is TRUE and to reset one or more bits in an integer-type data if the condition is FALSE.

Syntax	**i32 wbit (i32 data, i32 bitMsk, bool cond)**
data	Unit to which the write is desired
bitMsk	Mask of the bits to be written
cond	Condition upon the occurrence of which the bits will be set/reset
Result	Returns: Value: Meaning 1: The condition is verified 0: Otherwise
Validity	Rule, Task
Note	

© 2025 Robox SpA

Examples

Example of use:

```
sbit647(r(1), 0x000000ff, r(100) = 34)
; sets the low 8 bits to 1 when r(100) is equal to 34
; sets the low 8 bits to 0 when r(100) is different from 34
; equal to:
if(r(100) = 34))
    r(1) = r(1) r_or 0x000000ff
else
    r(1) = r(1) r_and r_not 0x000000ff
endif
```

Function wle2be()

Function that converts little endian word (x16 or x32) to big endian word (x16).

Syntax	**i16 wle2be (i16|i32 data)** **u16 wle2be (u16|u32 data)**
data	Data input
Result	Return value
Validity	Rule, Task
Note	

Examples

Example of use:

```
U16 input_1
U32 input_2
U16 result
input_1 = 0x0201
input_2 = 0x04030201

result = wle2be648(input_1)
; result = 0x0102
result = wle2be(input_2)
; result = 0x0102
```

Function write_dis()

Function that allows writing to a display.

Depending on the control used, writing occurs on different displays. With an RP-2 control this function writes to the 4x20 display built into the control. With other control models, writing takes place on the dispan module, if one is connected to the control.

© 2025 Robox SpA

Syntax	**i32 write_dis (string str [, i32 len=40] [, i32 offset=0])**
str	String to be displayed
len	Number of characters to display. (optional, default 40)
offset	Start of buffer. (optional, default 0)
Result	Returns:

Value	Meaning
<> 0	Successful writing
0	Failed

Validity	Rule, Task
Note	The last two lines of the display are handled by RTE via appropriate menus. Act on the predefined variable dis_display to limit/reduce these managements

Examples

Example of use:

```
; definitions of variables
STRING 899 stringaRiga1, stringaRiga2
; we write on the first two lines of the display date and time
stringaRiga1 = "DATA : " # 781 str_i(day, 2, 10) #"/"# str_i(month, 2,
10)  # "/"# str_i(year, 4, 10) # "    "
stringaRiga2 = "ORA  : " # str_i(hour, 2, 10)#"/"# str_i(minute, 2,
10) # "/"# str_i(second, 2, 10) # "      "
; I write the first string
write_dis 648 (stringaRiga1,20,0)
; I write the second string
write_dis(stringaRiga2,20,20)

; you want to disable the display from RTE and clean it up
dis_display = 0xf
; I write an empty string of 40 characters
write_dis("                                        ", 40, 0)
```

© 2025 Robox SpA

Instructions

Instruction __main_loop__

This Instruction can be used as an alternative to the "loop:" or "goto 693 loop" instructions to identify the main block of an R3 program.

Syntax	**__main_loop__**
Validity	Task
Note	If break 667 is present, no code is executed until end_main_loop 680. The behavior of the __main_loop__ and end_main_loop instructions are analogous to the while 760 and end_while 683 construct with condition always true

Examples

Example of use:

```
; I call an initialization routine
call 667 initParam
; start the main loop
__main_loop__
; main loop instruction
 ...
; I measure the duration of the main loop and store the value in the
variable T1
 T1 = loop_time 363()
; I run statistics on the duration of the main loop
 returns = statistics 595(T1, statT1)
; end of main loop --> return to beginning
end_main_loop
```

Example of use:

```
; I call an initialization routine
call 667 initParam
; start the main loop
__main_loop__
; main loop instruction
 ...
; if the value of the variable "finished" is equal to 1 I exit the
loop,
; then the instrictions from this point to the end of the loop will
not be executed
 _if 651 (finished)
 break 667
; other main loop instructions
 ...
; end of main loop --> return to beginning
end_main_loop 680
```

© 2025 Robox SpA

```
; other instructions outside the main loop
```

Instruction _else

This Instruction can be used only in conjunction with the _if block whose it identifies the alternative.

Syntax	**_else**
Validity	Rule, Task
Note	Just as is the case with the _if instruction, _else it conditions only the next line. In case the user enters a second instruction in the branch it will always be executed (even if the _else condition is false)! This Instruction should never be followed by: _if, _else, if, do, for, select, while, __main_loop__

Example

```
; If the axis control has been on for more than 60 seconds (rising
edge of the condition).
 _if 651 (tfb > 60)
; then in the string register sr(1) will be stored the words.
; " I have been on for more than one minute "
 sr(1) = " I have been on for more than one minute "
; otherwise in the string register sr(1) will be stored the
inscription
; " I have been on for less than a minute "
_else
 sr(1) = " I have been on for less than a minute "
```

Instruction _if

This instruction conditions the execution of only the next line, which is executed only if the condition is true.

Syntax	**_if**
Validity	Rule, Task
Note	In case the user enters a second Instruction in the branch it will always be executed (even if the _if condition is false)! It can be used alone or with _else (to handle the false condition as well).

© 2025 Robox SpA

Syntax	**_if**
	This Instruction should never be followed by: _if, _else, if, do, for, select, while, __main_loop__

Examples

Example usage:

```
; on the rising edge of digital input 257
_if(rise 536(inp(257)))
; we set the digital output 257
 out(257) = 1
; on the rising edge of digital input 258
_if(rise 536(inp(258)))
; we reset the digital output 257
 out(257) = 0
```

Example usage:

```
; If the motion control has been on for more than 60 seconds (rising
edge of condition).
_if(tfb > 60)
; then in the string register sr(1) will be stored the words.
; " I have been on for more than one minute "
 sr(1) = " I have been on for more than one minute "
; otherwise in the string register sr(1) will be stored the
inscription
; " I have been on for less than a minute "
_else 651
 sr(1) = " I have been on for less than a minute "
```

Instruction aux

Start of aux block within a rule. The block must end with the end_aux instruction.

Syntax	**aux**
Validity	Rule
Notes	In the aux end_aux block, the programmer should enter handles detached from axis motion (this block can be omitted)

Examples

Example of use:

```
; beginning of Rule
rule 720 1
 ...
```

```
 ; block ref (optional)
 ref 717
 ...
 ; end block ref
 end_ref 681
 ; motion block (required)
 motion 703
 ...
 ; end motion block
 end_motion 681
 ; aux block (optional)
 aux
 ...
 ; activate output 250 to signal that I am in this block (as an
example)
 OUT(250) = 1
 ...
 ; end aux block
 end_aux 678
 ...
; end of rule
end_rule 682
```

Instruction axes

Definition of axes involved within a rule 720.

Syntax	**axes (i32 nax1, i32 nax2, ..., i32 naxn)**
nax1	Number of the first axis involved
nax2	Number of the second axis involved
naxn	Number of the nth axis involved
Validity	Rule
Note	

Examples

Example of use:

```
; the first three axes are involved in the rule
axes(1, 2, 3)
```

Note: The above example is equivalent to: axes_m 654 (0x7).

Instruction axes_m

Definition of axes involved within a rule expressed through a mask.

Syntax	**axes_m (i32 axmask)**
axmask	Mask of the axes involved
Validity	Rule
Note	

Examples

Example of use:

```
; the first 12 axes are involved in the rule
axes_m(0xfff)
```

Note: The example above is equivalent to: axes 653 (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)

Instruction b0

This Instruction is used to write byte 0 of the argument.

Syntax	**b0 (i8|i16|i32|u8|u16|u32|real arg) = u8 value**
arg	Variable in which the *value* will be imposed
value	Value to be imposed
Validity	Rule, Task
Note	From RTE 33.9.0 accessing a non-existing byte returns error

Instruction b1

This Instruction is used to write byte 1 of the argument.

Syntax	**b1 (i16|i32|u16|u32|real arg) = u8 value**
arg	Variable in which the *value* will be imposed
value	Value to be imposed
Validity	Rule, Task
Note	From RTE 33.9.0 accessing a non-existing byte returns error

© 2025 Robox SpA

Instruction b2

This Instruction is used to write byte 2 of the argument.

Syntax	**b2 (i32|u32|real arg) = u8 value**
arg	Variable in which the *value* value will be imposed
value	Value to be imposed
Validity	Rule, Task
Note	As of RTE 33.9.0 access to a non-existing byte returns error

Instruction b3

This Instruction is used to write byte 3 of the argument.

Syntax	**b3 (i32|u32|real arg) = u8 value**
arg	Variable in which the *value* value will be imposed
value	Value to be imposed
Validity	Rule, Task
Notes	As of RTE 33.9.0 access to a non-existing byte returns error

Instruction b4

This Instruction is used to write byte 4 of the argument.

Syntax	**b4 (real arg) = u8 value**
arg	Variable in which the *value* value will be imposed
value	Value to be imposed
Validity	Rule, Task
Note	As of RTE 33.9.0 access to a non-existing byte returns error

Instruction b5

This Instruction is used to write byte 5 of the argument.

Syntax	**b5 (real arg) = u8 value**
arg	Variable in which the *value* value will be imposed

© 2025 Robox SpA

Syntax	**b5 (real arg) = u8 value**
value	Value to be imposed
Validity	Rule, Task
Note	As of RTE 33.9.0 access to a non-existing byte returns error

Instruction b6

This Instruction is used to write byte 6 of the argument.

Syntax	**b6 (real arg) = u8 value**
arg	Variable in which the *value* value will be imposed
value	Value to be imposed
Validity	Rule, Task
Note	As of RTE 33.9.0 access to a non-existing byte returns error

Instruction b7

This Instruction is used to write byte 7 of the argument.

Syntax	**b7 (real arg) = u8 value**
arg	Variable in which the *value* value will be imposed
value	Value to be imposed
Validity	Rule, Task
Note	As of RTE 33.9.0 access to a non-existing byte returns error

Instruction bcc_set_double

This function allows a REAL to be written to the *data* field of a BCC_MSGUSER 793 structure at the specified offset.

Syntax	**bcc_set_double (bcc_msguser str, int offset, real value)**
str	Structure of type bcc_msguser 793
offset	Offset in number of bytes you want to write to
value	The value you want to enter

© 2025 Robox SpA

Syntax	**bcc_set_double (bcc_msguser str, int offset, real value)**
Validity	Rule, Task
Note	

Examples

Example of use:

```
bcc_msguser bcc_str
Real valueToSend = 3.14
Int offset = 10

bcc_set_double(bcc_str, offset, valueToSend)
```

Instruction bcc_set_float

This function allows a FLOAT to be written to the *data* field of a BCC_MSGUSER 793 structure at the specified offset.

Syntax	**bcc_set_double (bcc_msguser str, int offset, float value)**
str	Structure of type bcc_msguser 793
offset	Offset in number of bytes to be written to
value	The value you want to enter
Validity	Rule, Task
Note	

Examples

Example of use:

```
bcc_msguser bcc_str
float valueToSend = 3.14
int offset = 10

bcc_set_float(bcc_str, offset, valueToSend)
```

© 2025 Robox SpA

Instruction bcc_set_i16

This function allows an i16 to be written to the *data* field of a BCC_MSGUSER 793 structure at the specified offset.

Syntax	**bcc_set_double (bcc_msguser str, int offset, i16 value)**
str	Structure of type bcc_msguser 793
offset	Offset in number of bytes to be written to
value	The value you want to enter
Validity	Rule, Task
Note	

Examples

Example of use:

```
bcc_msguser bcc_str
i16 valueToSend = 0x1234
int offset = 10

bcc_set_i16(bcc_str, offset, valueToSend)
```

Instruction bcc_set_i32

This function allows an i32 to be written to the *data* field of a BCC_MSGUSER 793 structure at the specified offset.

Syntax	**bcc_set_double (bcc_msguser str, int offset, i32 value)**
str	Structure of type bcc_msguser 793
offset	Offset in number of bytes to be written to
value	The value you want to enter
Validity	Rule, Task
Note	

Examples

Example of use:

```
bcc_msguser bcc_str
i32 valueToSend = 0x12345678
int offset = 10

bcc_set_i32(bcc_str, offset, valueToSend)
```

© 2025 Robox SpA

Instruction bcc_set_i8

This function allows an i8 to be written to the *data* field of a BCC_MSGUSER 793 structure at the specified offset.

Syntax	**bcc_set_double (bcc_msguser str, int offset, i8 value)**
str	Structure of type bcc_msguser 793
offset	Offset in number of bytes to be written to
value	The value you want to enter
Validity	Rule, Task
Note	

Examples

Example of use:

```
bcc_msguser bcc_str
i8 valueToSend = 0x12
int offset = 10

bcc_set_i8(bcc_str, offset, valueToSend)
```

Instruction bcc_set_string

This function allows a STRING to be written to the *data* field of a BCC_MSGUSER 793 structure at the specified offset.

Syntax	**bcc_set_double (bcc_msguser str, int offset, string value)**
str	Structure of type bcc_msguser 793
offset	Offset in number of bytes to be written to
value	The value you want to enter
Validity	Rule, Task
Note	

Examples

Example of use:

```
bcc_msguser bcc_str
string valueToSend = "Hello world!"
int offset = 10
```

© 2025 Robox SpA

```
bcc_set_string(bcc_str, offset, valueToSend)
```

Instruction bcc_set_u16

This function allows a u16 to be written to the *data* field of a BCC_MSGUSER 793 structure at the specified offset.

Syntax	**bcc_set_double (bcc_msguser str, int offset, u16 value)**
str	Structure of type bcc_msguser 793
offset	Offset in number of bytes to be written to
value	The value you want to enter
Validity	Rule, Task
Note	

Examples

Example of use:

```
bcc_msguser bcc_str
u16 valueToSend = 0x1234
int offset = 10

bcc_set_u16(bcc_str, offset, valueToSend)
```

Instruction bcc_set_u32

This function allows a u32 to be written to the *data* field of a BCC_MSGUSER 793 structure at the specified offset.

Syntax	**bcc_set_double (bcc_msguser str, int offset, u32 value)**
str	Structure of type bcc_msguser 793
offset	Offset in number of bytes to be written to
value	The value you want to enter
Validity	Rule, Task
Note	

Examples

Example of use:

```
bcc_msguser bcc_str
```

```
u32 valueToSend = 0x12345678
int offset = 10

bcc_set_u32(bcc_str, offset, valueToSend)
```

Instruction bcc_set_u8

This function allows a u8 to be written to the *data* field of a BCC_MSGUSER 793 structure at the specified offset.

Syntax	**bcc_set_double (bcc_msguser str, int offset, u8 value)**
str	Structure of type bcc_msguser 793
offset	Offset in number of bytes to be written to
value	The value you want to enter
Validity	Rule, Task
Note	

Examples

Example of use:

```
bcc_msguser bcc_str
u8 valueToSend = 0x12
int offset = 10

bcc_set_u8(bcc_str, offset, valueToSend)
```

Instruction bit_mask_add

Allows a value expressed in hexadecimal to be added to a variable of type I32.

Syntax	**bit_mask_add name value**
name	Name of the int variable to sum the *value* to
value	Value added to the variable. Must be written in hexadecimal, with "0x" in front
Note	WARNING: This Instruction is handled directly by the RDE environment. It is not recommended to use it from task or rule.

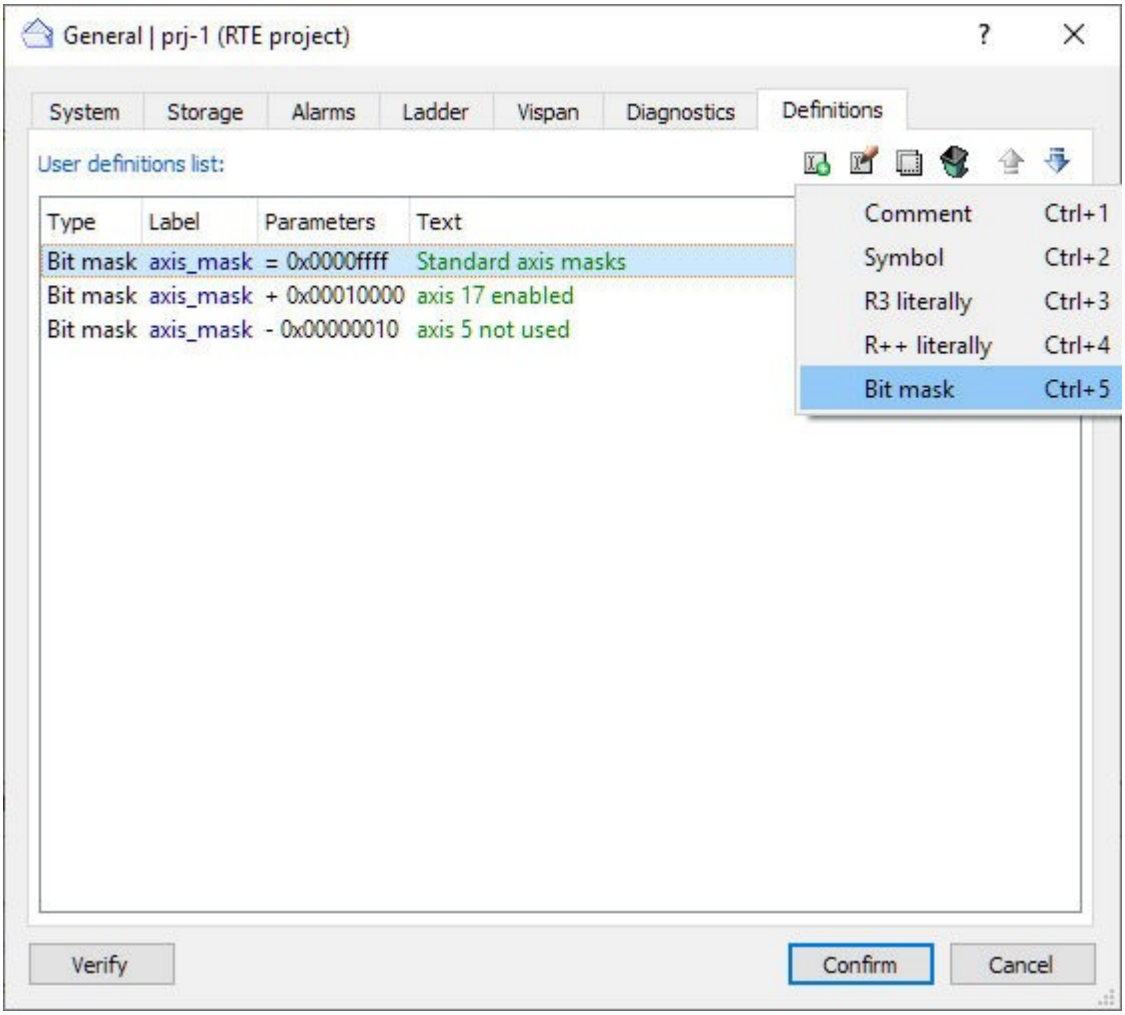
Examples

Example of use:

© 2025 Robox SpA

From RDE project -> General -> Definitions -> Bit mask, we add these three definitions:

- axis_mask, impose value 0x0000FFFF
- axis_mask, add bit 0x00010000
- axis_mask, remove bit 0x000010



Once the configuration is generated, in the file cfgdef.i3 we will have:

```
; Standard axis masks
bit_mask_set axis_mask 0x0000ffff
; axis 17 enabled
bit_mask_add axis_mask 0x00010000
; axis 5 not used
bit_mask_remove axis_mask 0x00000010
```

From the program we then have available the variable axis_mask of type INT, initialized to the value 0x0001FFEF

```
; task:
if(axis_mask.4)
    ...
endif
```

Instruction bit_mask_remove

Allows a value expressed in hexadecimal to be subtracted from a variable of type INT.

Sintassi	**bit_mask_remove name value**
name	Name of the variable to subtract the *value* to
value	Value subtracted from the variable. Must be written in hexadecimal, with "0x" in front
Note	**WARNING**: This Instruction is handled directly by the RDE environment. It is not recommended to use it from task or rule.

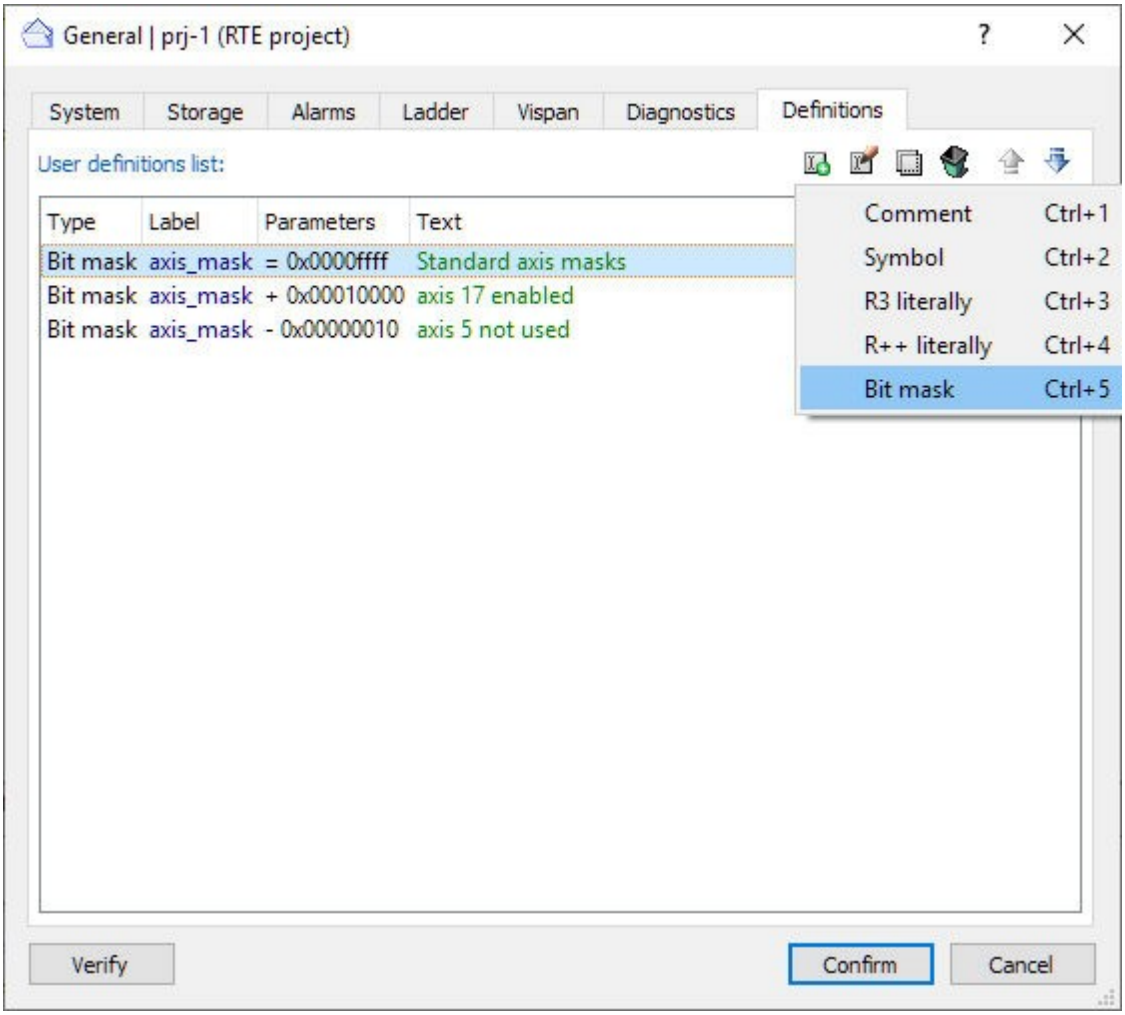
Examples

Example usage:

From RDE project -> General -> Definitions -> Bit mask, we add these three definitions:

- axis_mask, impose value 0x0000FFFF
- axis_mask, add bit 0x00010000
- axis_mask, remove bit 0x000010

© 2025 Robox SpA



Once the configuration is generated, in the file cfgdef.i3 we will have:

```
; Standard axis masks
bit_mask_set axis_mask 0x0000ffff
; axis 17 enabled
bit_mask_add axis_mask 0x00010000
; axis 5 not used
bit_mask_remove axis_mask 0x000010
```

From the program we then have available the variable axis_mask of type INT, initialized to the value 0x0001FFEF

```
; task:
if (axis_mask.4)
 ....
endif
```

© 2025 Robox SpA

Instruction bit_mask_set

Lets you initialize a variable of type INT to a value expressed in hexadecimal.

Syntax	**bit_mask_set name value**
name	Name of the INT variable that will be initialized
value	Value to which the variable is initialized. Must be written in hexadecimal, with "0x" in front
Note	**WARNING**: This Instruction is handled directly by the RDE environment. It is not recommended to use it from task or rule.

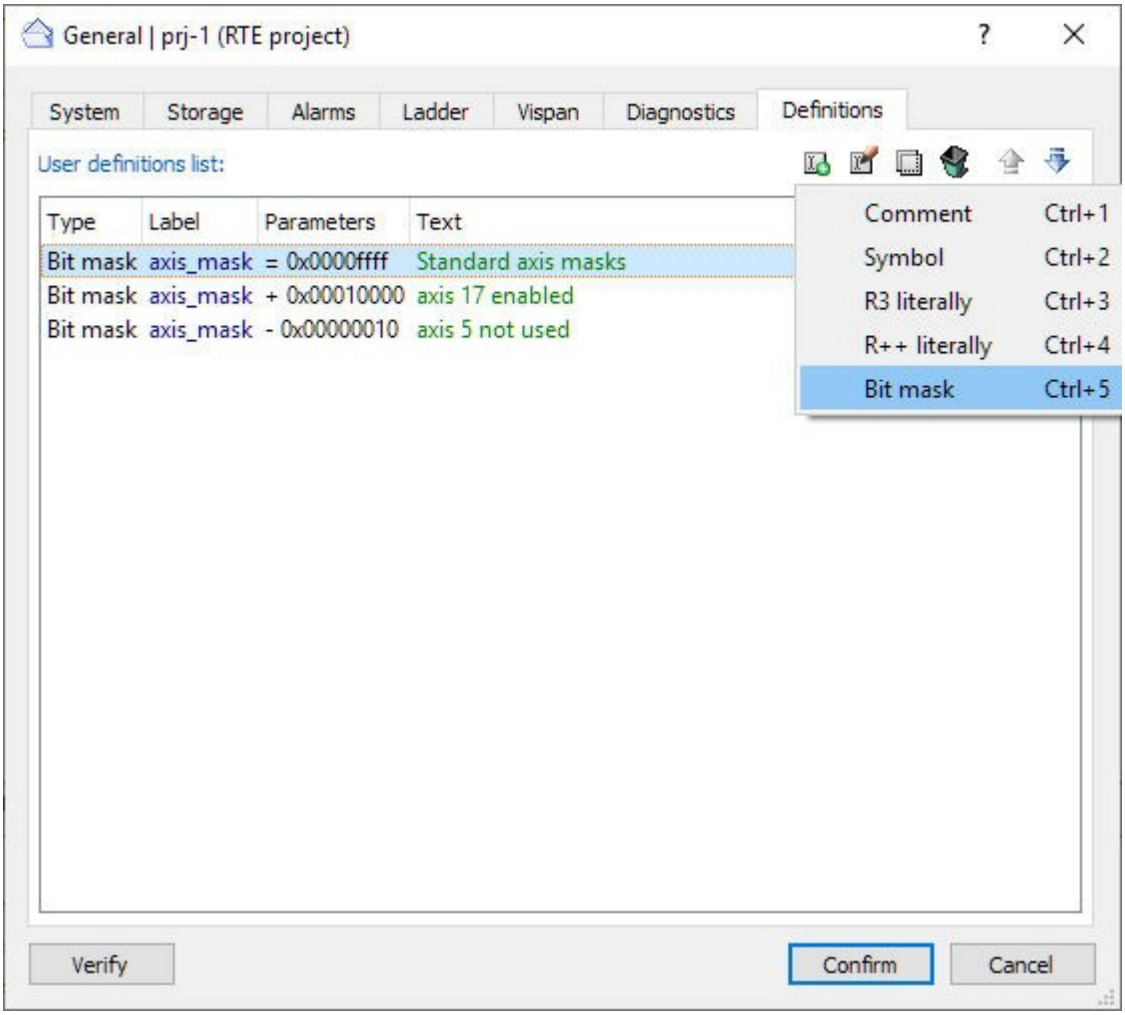
Examples

Example usage:

From RDE project -> General -> Definitions -> Bit mask, add these three definitions:

- axis_mask, impose value 0x0000FFFF
- axis_mask, add bit 0x00010000
- axis_mask, remove bit 0x000010

© 2025 Robox SpA



Once the configuration is generated, in the file cfgdef.i3 we will have:

```
; Standard axis masks
bit_mask_set axis_mask 0x0000ffff
; axis 17 enabled
bit_mask_add axis_mask 0x00010000
; axis 5 not used
bit_mask_remove axis_mask 0x000010
```

From the program we then have available the variable axis_mask of type INT, initialized to the value 0x0001FFEF

```
; task:
if (axis_mask.4)
 ....
endif
```

© 2025 Robox SpA

Instruction break

Instruction to exit a selection block.

Syntax	**break**
Validity	Rule, Task
Note	

Examples

Example of use:

```
; selection based on the variable pippo
select 736 (pippo)
    ; case pippo = 1
    case 669 1
        ...
        break
    ; case pippo = 2
    case 2
        ...
        break
    ; if no case above occurred
    default 675
        ...
end_select 682
```

Instruction call

Instruction call without passing parameters

This Instruction causes execution of the specified subprogram, after which, execution resumes from the next instruction.

Syntax	**call name()**
name	It is the name of the function 688 to be executed
Validity	Rule, Task
Note	Brackets used for parameter delimitation may be omitted, but only in case the function has no incoming parameters

Examples

Example of use:

```
$task1
; function call
```

© 2025 Robox SpA

```
call inizializzaVariabili()
; main loop of the program
__main_loop__
 ; program instructions
    ...
end_main_loop
;********************
; Function inizializzaVariabili
function 688 inizializzaVariabili
    pippo = 0
    pluto = 1
    ...
end_fun 680
```

Instruction call with parameter passing

Causes execution of the specified subprogram by providing it with the specified parameters, after which, execution resumes from the next Instruction.

Syntax	**call name (val1, val2, ... , valn)**
name	Is the name of the function 688 to be executed
val1	Is the value of the first parameter passed to the function
val2	Is the value of the second parameter passed to the function
valn	Is the value of the nth parameter passed to the function
Validity	Rule, Task
Note	The data types of the parameters passed to function must match those declared in it. In the case of passing a real value to a function in which a parameter of type integer has been declared, there will be an automatic type conversion, from real to integer, without error reporting

Examples

Example of use:

```
; function call without using the return value
call calc_diag1(12,17)
; function call inside a mathematical expression.
; uses the return value
rr(1) = 5.0 + calc_diag2(12,17)
...
; function calc_diag1
```

© 2025 Robox SpA

```
function 688 calc_diag1(REAL val1, REAL val2)
    if 696((val1<0) or (val2<0))
        rr(1) = 0
        return 719
    endif 683
    rr(1) = sqrt 594(val1*val1 + val2*val2)
end_fun 680

; function calc_diag2
function 688 REAL 898 calc_diag2(REAL val1, REAL val2)
    if 696((val1<0) or (val2<0))
        return 719 0.0
    endif 683
    return 719 sqrt(val1*val1 + val2*val2)
end_fun 680
```

Instruction case

Start of a group of instructions inside the select selection block.

If the select variable takes a value equal to *k*, it allows the instructions in the case to be executed.

Syntax	**case k**
k	Constant value
Validity	Rule, Task
Note	The given type of constant k must match the given type of the compared variable

Examples

Example of use:

```
; selection based on the variable pippo
select 736(pippo)
    ; case pippo = 1
    case 1
        ...
        break 667
    ; case pippo = 2
    case 2
        ...
        break
    ; if no case above occurred
    default 675
        ...
end_select 682
```

© 2025 Robox SpA

Instruction continue

Instruction that forces the jump to handle reiterations in a for, while or do loop.

Syntax	**continue**
Validity	Rule, Task
Note	

Examples

Example of use:

```
for 686 (i=0, i<10 , inc(i))
    if 696 (i<5)
        ...
        ; We reiterate the for loop, moving to the next index
        continue
    endif 683
    ; so here I get there only if i >= 5
    ...
end_for 679
```

Instruction ct_cd

Instruction that set an input to decrement a counter.

Syntax	**ct_cd (counter cnt, inCond)**
cnt	Name of counter 892 (defined as type of variable)
inCond	Count input (rising edge). Can be any condition, even an expression (e.g., cp(1) > 1000)
Validity	Rule, Task
Note	Counters are not retentive

On the rising edge of the input(*inCond*), the current value (CV) is decremented by 1. The output (QU) goes into the ON state when the current value (CV) becomes <= of the preset value (PV). The count (CV) is a non limited i32 type variable (from the maximum negative value goes to the maximum positive value).

Examples

Example of use:

```
counter 892 C1

ct_pv 675(C1, 10)        ; we set the preset to 10
ct_cu 672(C1, inp(1))    ; Up with input 1 (ADV button on CPU) ...
```

© 2025 Robox SpA

```
ct_cd 670 (C1, inp(2))    ; Down with input 2 (MODE button on CPU) ...

out(257) = ct_qu 271 (C1) ; We output out 257 if current value >= preset
out(258) = ct_qd 270 (C1) ; We output out 258 if current value <= 0
currVal  = ct_cv 269 (C1) ; Totalizer

_if(inp(3))
    ct_load 674 (C1)      ; loads the current value with the preset
value and CT_QU goes ON
_if(inp(4))
    ct_reset 675 (C1)     ; resets the current value to zero and CT_QD
goes to ON

; inp 1 = ADV ; inp 2 = MODE ; inp 3 = feed- ; inp 4 = feed+
```

Instruction ct_cdr

Instruction that set an input to decrement a circular counter.

Syntax	**ct_cdr (counter cnt, inCond)**
cnt	Counter 892 name (defined as type of variable)
inCond	Count input (rising edge). Can be any condition, even an expression (e.g., cp(1) > 1000)
Validity	Rule, Task
Note	Counters are not retentive

On the rising edge of the input(*inCond*), the current value (CV) is decremented by 1. The output (QU) goes into the ON state when the current value (CV) becomes <= of the preset value (PV). The count (CV) is a variable of type i32 that arrived at value 0 is reset to the preset value (PV).

Examples

Example of use:

```
counter 892 C1

ct_pv 675 (C1, 10)        ; we set the preset to 10
ct_cur 673 (C1, inp(1))    ; Up with input 1 (ADV button on CPU) ...
ct_cdr 671 (C1, inp(2)) ; Down with input 2 (MODE button on CPU) ...

out(257) = ct_qu 271 (C1) ; output out 257 if current value >= preset
out(258) = ct_qd 270 (C1) ; we output  out 258 if current value <= 0
currVal = ct_cv 269 (C1) ;  totalizer
```

© 2025 Robox SpA

```
_if(inp(3))
 ct_load(C1) ;  loads current value with preset value and CT_QU
goes ON
_if(inp(4))
 ct_reset(C1) ;  resets the current value to zero and CT_QD goes ON
```

Instruction ct_cu

Instruction that set an input to increment a counter.

Syntax	**ct_cu (counter cnt, inCond)**
cnt	Counter name (defined as type of variable)
inCond	Count input (rising edge). Can be any condition, even an expression (e.g., cp(1) > 1000)
Validity	Rule, Task
Note	A ct_cd counter can also act on the same counter. Counters are not retentive

On the rising edge of the input(*inCond*), the current value (CV) is incremented by 1. The output (QU) goes into the ON state when the current value (CV) becomes >= of the preset value (PV).

The count (CV) is an unrestricted i32 type variable (from the maximum positive value goes to the maximum negative value).

Diagnostics

Alarm 9901 ill.Fun is generated in case of using illegal parameters.

Examples

Example of use:

```
counter C1

ct_pv(C1, 10) ; we set the preset to 10
ct_cu(C1, inp(1)) ; Up with input 1 (ADV button on CPU) ...
ct_cd(C1, inp(2)) ; Down with input 2 (MODE button on CPU) ...

out(257) = ct_qu(C1) ; output out 257 if current value >= preset
out(258) = ct_qd(C1) ; we output  out 258 if current value <= 0
currVal = ct_cv(C1) ;  totalizer
```

```
_if(inp(3))
 ct_load(C1) ;  loads current value with preset value and CT_QU
goes ON
_if(inp(4))
 ct_reset(C1) ;  resets the current value to zero and CT_QD goes ON

; inp 1 = ADV ; inp 2 = MODE ; inp 3 = feed- ; inp 4 = feed+
```

Instruction ct_cur

Instruction that set an input to increment a circular counter.

Syntax	**ct_cur (counter cnt, inCond)**
cnt	Counter name (defined as type of variable)
inCond	Count input (rising edge). Can be any condition, even an expression (e.g., cp(1) > 1000)
Validity	Rule, Task
Note	A ct_cdr counter can also act on the same counter. Counters are not retentive

On the rising edge of the input(*inCond*), the current value (CV) is incremented by 1. The output (QU) goes into the ON state when the current value (CV) becomes > of the preset value (PV).

The count (CV) and a variable of type i32 which once it reaches the preset value (PV) is reset to zero.

Diagnostics

Alarm 9901 ill.Fun is generated if illegal parameters are used.

Examples

Example of use:

```
counter C1

ct_pv(C1, 10) ; we set the preset to 10
ct_cur(C1, inp(1)) ; Up with input 1 (ADV button on CPU) ...
ct_cdr(C1, inp(2)) ; Down with input 2 (MODE button on CPU) ...

out(257) = ct_qu(C1) ; output out 257 if current value >= preset
out(258) = ct_qd(C1) ; we output  out 258 if current value <= 0
currVal = ct_cv(C1) ;  totalizer
```

© 2025 Robox SpA

```
_if(inp(3))
 ct_load(C1) ;  loads current value with preset value and CT_QU
goes ON
_if(inp(4))
 ct_reset(C1) ;  resets the current value to zero and CT_QD goes ON
```

Instruction ct_cv

Instruction to impose the current value of the counter.

NOTE: The similar function ct_cv() is used instead to read the current value of the counter.

Syntax	**ct_cv (counter cnt, real value)**
cnt	Name of the counter (defined as a variable type)
value	Current value to be imposed
Validity	Rule, Task
Note	Useful for creating retentive counters

Examples

Example usage:

```
ct_cv(C1, rr(1000)) ; I write the value contained in the real
register rr(1000) into the current value of counter C1
currVal = ct_cv(C1) ; I write the current value of the counter into
the variable currVal
```

Instruction ct_load

Instruction to load the current value (CV) of the counter with the preset value (CP).

Syntax	**ct_load (counter cnt)**
cnt	Name of counter (defined as type of variable)
Validity	Rule, Task
Note	

© 2025 Robox SpA

Instruction ct_pv

Instruction to impose the counter preset value.

NOTE: The analogous function ct_pv 270() is used instead to read the counter preset value.

Syntax	**ct_pv (counter cnt, real preset)**
cnt	Name of the counter 892 (defined as a variable type)
preset	Value of the preset
Validity	Rule, Task
Note	

Examples

Example usage:

```
ct_pv 675(C1, rr(1000)) ; I write the value contained in the real register rr(1000) into the preset of counter C1
presetVal = ct_pv(C1) ; I write the value of the preset into the variable presetVal
```

Instruction ct_reset

Instruction to reset the counter of the specified counter to 0.

Syntax	**ct_reset (counter cnt)**
cnt	Name of the counter 892 (defined as a variable type)
Validity	Rule, Task
Note	

Instruction default

Start of the default Instruction group inside the select block of the select construct.

Syntax	**default**
Validity	Rule, Task
Note	If no "case" can be associated with the selection variable, the "default" instruction group, if any, will be executed

Examples

Example usage:

© 2025 Robox SpA

```
; selection based on the variable pippo
select 736(pippo)
 ; case pippo = 1
 case 1
 ...
 break 667
 ; case pippo = 2
 case 2
 ...
 break
 ; if no case above occurred
 default
 ...
end_select 682
```

Instruction do

All instructions following this function will be repeated until the end_do_while 679 condition becomes false.

Syntax	**do**
Validity	Rule, Task
Note	

Examples

Example usage:

```
; we reset the first 100 real retentive registers to zero
Int i = 0
do
 i = i+1
 nvrr(i) = 0
end_do_while 679(i <= 100)
```

Instruction dw0

This Instruction is used to write the first dword (starting with the least significant) of the argument.

Syntax	**dw0 (i32|u32|real arg) = u32 val**
arg	Arg of the first dword
value	Value to be imposed in *arg*
Validity	Rule, Task
Note	Since RTE 33.9.0 accessing a non-existing dword returns an error

© 2025 Robox SpA

Instruction dw1

This Instruction is used to write the second dword (starting with the least significant one) of the argument.

Syntax	**dw1 (i32|u32|real arg) = u32 val**
arg	Arg of the second dword
value	Value to be imposed in *arg*
Validity	Rule, Task
Note	Since RTE 33.9.0 accessing a non-existing dword returns an error

Instruction dwell

Suspends task execution for the specified number of seconds.

Syntax	**dwell (real val)**
val	Number of seconds to wait. If a negative number is used, the Instruction will not suspend program execution (equivalent to dwell(0))
Validity	Task
Note	

Examples

Example of use:

```
; the task is suspended for 5 seconds
dwell(5)
; the task is suspended for half a second
dwell(0.5)
; the task is suspended for a number of seconds equal to the value
; contained in the real register rr(23)
dwell(rr(23))
```

Instruction else

Start of a group of alternative choice instructions in the if 696 - else 677 - endif 683 construct.

Syntax	**else**
Validity	Rule, Task
Note	

© 2025 Robox SpA

Examples

Example usage:

```
; if the contents of the real register rr(2) is equal to zero.
if (rr(2) = 0)
 ; then the content of the real register rr(1) is multiplied by 2
 rr(1 ) =rr(1) * 2
; otherwise the contents of the real register rr(1) is divided by 2
else
 rr(1) = rr(1) / 2
 ; end of if_else_endif construct.
endif
```

Instruction end_aux

End of aux block within the rule.

Syntax	**end_aux**
Validity	Rule
Note	

Examples

Example of use:

```
; beginning of Rule
rule 1
 ...
 ; block ref (optional)
 ref
 ...
 ; end block ref
 end_ref
 ; motion block (required)
 motion
 ...
 ; end motion block
 end_motion
 ; aux block (optional)
 aux
 ...
 ; activate output 250 to signal that I am in this block (as an
example)
 OUT(250) = 1
 ...
 ; end aux block
 end_aux
 ...
```

© 2025 Robox SpA

```
; end of rule
end_rule 682
```

Instruction end_do_while

End of the do construct.

Syntax	**end_do_while**
Validity	Rule, Task
Note	

Examples

Example usage:

```
; we reset the first 100 real retentive registers to zero.
Int i = 0
do 676
 inc(i)
 nvrr(i) = 0
end_do_while(i < 100)
```

Instruction end_for

End of the for construct.

Syntax	**end_for**
Validity	Rule, Task
Note	

Examples

Example usage:

```
; for construct
for 686(i = 0, i < 22, i=i+2)
 pippo = pippo + i
; end of for construct for
end_for
```

© 2025 Robox SpA

Instruction end_fun

End of instruction function 688().

Syntax	**end_fun**
Validity	Rule, Task
Note	

Examples

Example of use:

```
; ***********************************
; * Function initializes variables *
; ***********************************
function 688 initializeVariables
 pippo = 0
 pluto = 1
 ...
end_fun
```

Instruction end_main_loop

This Instruction can be used as an alternative to the loop: or goto 693 loop instruction to identify the main block of an R3 program.

Syntax	**end_main_loop**
Validity	Task
Note	If break 667 present, no code is executed until end_main_loop. The behavior of the __main_loop__ / end_main_loop instructions are analogous to the while / end_while construct with condition always true

Examples

Example usage:

```
; start of the main loop of the program (main loop).
__main_loop__ 650
 ; Instruction of the main loop of the program (main loop)
 ...
; end of the main loop of the program (main loop)
end_main_loop
```

© 2025 Robox SpA

Instruction end_motion

End of block motion within a rule 720.

Syntax	**end_motion**
Validity	Rule
Note	

Examples

Example of use:

```
; beginning of rule
rule 720 1
 ; definition of axes involved
 axes 653 (1)
 ; start of motion block
 motion 703
 ; motion block instructions
 iv(1) = 0
 ; end of motion block
 end_motion
; end of rule
end rule 682
```

Instruction end_ref

End of ref block within a rule.

Syntax	**end_ref**
Validity	Rule
Note	

Examples

Example of use:

```
; beginning of rule
rule 720 1
 ; start of block ref
 ref 717
 ; ref block instruction
 ...
 ; end of ref block
 end_ref
 ; other blocks (motion, aux)
 ...
; end of rule
```

© 2025 Robox SpA

```
end_rule 682
```

Instruction end_rule

End of block rule within the motion 703 task.

Syntax	**end_rule**
Validity	Rule
Note	

Examples

Example of use:

```
; beginning of rule
rule 720 1
 ; definition of axes involved
 axes 653 (1)
 ; ref block (optional)
 ref 717
 ; ref block instructions
 ...
 ; end of ref block
 end_ref 681
 ; motion block (required)
 motion 703
 ; motion block instructions
 ...
 ; end of motion block
 end_motion 681
 ; aux block (optional)
 aux 652
 ; aux block instructions
 ...
 ; end of aux block
 end_aux 678
; end of rule
end_rule
```

Instruction end_select

End of the select construct.

Syntax	**end_select**
Validity	Rule, Task
Note	

© 2025 Robox SpA

Examples

Example usage:

```
; selection based on the variable pippo
select 736 (pippo)
 ; case pippo = 1
 case 669 1
 ...
 break 667
 ; case pippo = 2
 case 2
 ...
 break
 ; if no case above occurred
 default 675
 ...
end_select
```

Instruction end_while

End of the while construct.

Syntax	**end_while**
Validity	Rule, Task
Note	

Examples

Example usage:

```
; beginning of the while construct
while 760 (inp(1))
 ; Instruction R3 internal to the while construct.
 ...
; end of while construct
end_while
```

Instruction endif

End of the if construct.

Syntax	**endif**
Validity	Rule, Task
Note	

© 2025 Robox SpA

Examples

Example usage:

```
; if the contents of the real register rr(2) is equal to zero
if 696 (rr(2) = 0)
 ; then the content of the real register rr(1) is multiplied by 2
 rr(1) = rr(1) * 2
; otherwise the contents of the real register rr(1) is divided by 2
else 677
 rr(1) = rr(1) / 2
; end of if_else_endif construct.
endif
```

Instruction excl_lam

Instruction to include/exclude powerset logical axes.

Syntax	**powersetname.EXCL_LAM = axMask**
powersetname	Name of powerset
axMask	Mask of logical axes to include/exclude (1 exclude, 0 include)
Validity	Rule, Task
Note	If the powerset is in power, the operation is aborted and alarm 9904 is generated. If one of the axes to be included is already used in another powerset, the operation is aborted and alarm 9909 is generated

Examples

Example of use:

Into cfgdefs.i3:

```
...
; {7:0} Power set variables
...
power_set sys
power_set ps_qua          ; powerset definition
...
```

Into task or rule:

```
INT maskAx                ; axes mask excluded/reincluded
...
ps_qua.EXCL_LAM = maskAx
```

© 2025 Robox SpA

Instruction excl_pam

Instruction to include/exclude the physical axes of the powerset.

Syntax	**powersetname.EXCL_PAM = axMask**
powersetname	Powerset name
axMask	Mask of physical axes to include/exclude (1 exclude, 0 include)
Validity	Rule, Task
Notes	If the powerset is in power, the operation is aborted and alarm 9904 is generated. If one of the axes to be reincluded is already used in another powerset, the operation is aborted and alarm 9909 is generated

Examples

Example usage:

Into cfgdefs.i3:

```
...
; {7:0} Power set variables
...
power_set sys
power_set ps_qua ; powerset definition
...
```

Into task or rule:

```
INT maskAx ; axes mask excluded/reincluded
...
ps_qua.EXCL_PAM = maskAx
```

Instruction fill_mem

This Instruction allows you to initialize a variable, usually of indexed type, by assigning it a specified value repeated for a specified number of times.

Syntax	**fill_mem (varDest, nEl, val)**
varDest	Is the variable to be initialized. Can be a local variable or an integer register, REAL or STRING
nEl	Is the number of elements to be initialized, REAL or I32. It can also be a mathematical expression
val	Is the value used in the initialization, REAL or I32. It can also be a mathematical expression

© 2025 Robox SpA

Syntax	**fill_mem (varDest, nEl, val)**
Validità	Rule, Task
Note	Nel caso di registri, l'inizializzazione viene estesa ai registri successivi a quello indicato. Nel caso delle variabili locali, che avranno tipicamente almeno un indice, occorre fare attenzione a non tentare di inizializzare un numero di elementi superiore alla dimensione dichiarata della variabile stessa. In caso contrario si avrà un opportuno messaggio di errore in fase di esecuzione del programma. Questo controllo è eseguito solamente nel caso in cui *varDest* sia una variabile locale. Il numero *nEl* non rappresenta un numero di byte, ma un numero di elementi; ad es. nel caso di variabili reali, se si specifica *nEl* = 5 verranno inizializzate cinque locazioni reali consecutive della variabile specificata (in totale 40 bytes)

NOTE: When initializing integer registers or integer variables, *val* must always equal zero.

Examples

Example with integer registers:

```
; integer registers 1000 to 1099 are initialized to 0
fill_mem(r(1000), 100, 0)
```

Example with local array:

```
; the first 20 locations of the one-dimensional array pluto are
initialized with the value 12.34
REAL 898 pluto [40]
fill_mem(pluto, 20, 12.34)
```

Example with string:

```
; The first 40 characters of the message string are initialized with
the character "space" (= 0x20)
STRING 899 message
fill_mem(message, 40, 0x20)
```

Instruction for

This Instruction allows the execution of a group of instructions to be repeated a defined number of times.

Syntax	**for (init, cond, update)**
init	Loop initialization expression, typically a variable assignment (optional, default 0)

© 2025 Robox SpA

Syntax	**for (init, cond, update)**
cond	Boolean expression describing the stay condition within the FOR loop
update	Update expression, typically variable increment/decrement. (optional, default 1)
Validity	Rule, Task
Note	Contrary to what happens in C, in R3 the declaration of a variable inside the arguments of a for loop is not allowed. It is therefore necessary to declare the variable previously

Examples

Example of use:

```
Int i
...
for(i = 0, i < 22, i=i+2)
    pippo = pippo + i
end_for 679
```

In this example, the sequence of events is as follows:

1. The count variable i is initialized to zero,
2. The loop persistence condition "i < 22" is evaluated and recognized as true,
3. The instruction block contained in the FOR loop is executed (in this case, the single instruction "foo = foo + i"),
4. The variable "i" is incremented by two,
5. The loop starts over (step 3) and is executed a total of 11 times, i.e., as long as the loop condition remains true.

Note: In case the loop stay condition is immediately recognized as false, the FOR loop instruction block will never be executed.

Example of omitting optional data:

Using the FOR loop it is possible to omit both the initialization of the count variable and the increment of the count variable for example the following Instruction:

```
for( ,1 , )
    rr(14) = 0
    rr(15) = 0
end_for 679
```

They cause loop instructions to repeat indefinitely, the condition of remaining in loop is worth 1 and is always true.

Example of use with the continue instruction:

© 2025 Robox SpA

Use of the continue instruction within a FOR loop is possible; its execution causes the line in question to jump to the end_for instruction without executing any other instruction

```
for(i = 1, i < 7, i = i+1)
 if 696 (i = 3)
 ; skips the zeroing of r(3)
 continue 670
 endif 683
 r(i) = 0
end_for 679
```

Example of use with the break instruction:

The use of the break instruction within a FOR loop is possible; its execution causes the FOR loop to exit immediately without executing any other instructions.

```
for( ,1 , )
 ; permanence condition always true
 if 696(ip(3) > 100.3)
 ; exit the loop if ip(3)>100.3
 break 667
 endif 683
 ; count how many times the loop is executed
 rr(100) = rr(100)+1
end_for 679
```

Example with dynamic assignments:

The initial value of the FOR loop control variable is equal to the sum of the contents of the nonvolatile integer register nvr(12) and the nonvolatile integer register nvr(14). The loop is executed until j is less than the difference between the contents of the nonvolatile integer register nvr(22) and the nonvolatile integer register nvr(24). The variable j is incremented by 1 with each iteration.

```
for(j = nvr(12) + nvr(14), j < (nvr(22)-nvr(24)), inc 351(j))
     ...
end_for 679
```

Instruction function

Instruction that allows subprograms to be declared that can be called from anywhere in the user program.

The use of functions allows those instructions that must be executed repeatedly to be grouped into a single portion of code, with the aim of reducing memory occupancy and improving program clarity.

Simple function

Syntax	**function name ()** .... Subprogram Instruction ... **end_fun**
name	This is the identifier of the function and follows the normal rules 761 that apply to variable names
Validity	Rule, Task
Note	Use instruction call 667 to call the function

Function with parameter passing

Syntax	**function [type] name ([par1][, parN])** .... Subprogram instruction ... **[return** value **]** ; (only if return value present) **end_fun**
type	It is the eventual keyword i32 or real that defines the type of the parameter to be passed or the type of the return value. (optional)
name	It is the identifier of the function and follows the normal rules 761 that apply to variable names
par1	It is the value of the first parameter passed to the function
parN	It is the eventual value of the n-th parameter passed to the function. (optional)
Validity	Rule, Task
Note	Use instruction call 667 to call the function. When the function has no incoming parameter, brackets used for parameter delimitation may be omitted. In the case of passing a real value to a function where a parameter of type integer has been declared, there will be an automatic type conversion, from real to integer, without error reporting. Since RC3E 33.01.08, the passing of array-type parameters to functions (arrays of int, real, and structures) has been added. Structures passed to a function are always passed as references.

© 2025 Robox SpA

Syntax	**function [type] name ([par1][, parN])** .... Subprogram instruction ... **[return** value **]** ; (only if return value present) **end_fun**
	From RC3E 33.01.21 it is possible to pass RPE objects: (type Axes_Group, Point_l, Point_c, Point_j, Path, Transform)

Examples

Example of use:

```
; function call without using the return value
call calc_diag1(12,17)
; function call within a mathematical expression
rr(1) = 5.0 + calc_diag2(12,17)
; func calc_diag1
function calc_diag1 (REAL val1, REAL val2)
    if ((val1<0) or (val2<0))
        rr(1) = 0.0
    else
        rr(1) = sqrt(val1*val1 + val2*val2)
    endif
end_fun
; func calc_diag2
function REAL calc_diag2 (REAL val1, REAL val2)
    if ((val1<0) or (val2<0))
        return 0.0
    endif
    return sqrt(val1*val1 + val2*val2)
end_fun
```

Example of use:

```
STRUCT mys
    INT uno
    REAL due
END_STRUCT
mys s1
mys s2
mys s3

INT incs
INT myi1[5]
INT myi2[5]
myi1[0]= 1
myi1[1]= 2
myi1[2]= 3
```

© 2025 Robox SpA

```
myi1[3]= 4
myi1[4]= 5
; task 1
__MAIN_LOOP__
    IF (incs)
        incs = 0
        CALL f1(s1)
        CALL f2(s2)
        s3.uno = s3.uno+1
        s3.due = s3.due+1
        CALL f3(myi1, myi2)
    ENDIF
END_MAIN_LOOP
; functions
FUNCTION f1 (mys sf)
    sf.uno = sf.uno+1
    sf.due = sf.due+1
END_FUN
FUNCTION f2 (mys@ sf)
    sf.uno = sf.uno+1
    sf.due = sf.due+1
END_FUN
FUNCTION f3 (INT fi[5], INT fi2[5])
    fi2[0] = fi[0]
    fi2[1] = fi[1]
    fi2[2] = fi[2]
    fi2[3] = fi[3]
    fi2[4] = fi[4]
    fi[0] = fi[0] + 1
    fi[1] = fi[1] + 1
    fi[2] = fi[2] + 1
    fi[3] = fi[3] + 1
    fi[4] = fi[4] + 1
END_FUN
```

Example of use:

```
; storage
REAL localVar
struct s1
  real myVar
end_struct
s1 myStruct
; program
call incVariabile(localVar, -1)
call incVariabile(myStruct.myVar, 12.3)
; task 1
__MAIN_LOOP__
END_MAIN_LOOP
; reference
FUNCTION incVariabile(real@ varToInc , real increm)
     varToInc = varToInc + increm
```

© 2025 Robox SpA

```
END_FUN
```

Instruction get_i64

This function reads 8 bytes from a buffer, at the specified offset, interpreting them as a stru_large_int structure and writing the value read into the given structure.

The mirror function is set_i64 742().

Syntax	**get_i64 (buffer, i32 offset, stru_large_int data)**
buffer	Can be an array, structure, register
offset	Offset in bytes
data	Variable of type stru_large_int 844 in which to save the value read
Validity	Rule, Task
Note	

Examples

Once you have loaded the example program on a control, via a shell you can verify the functioning of get_i64. To do this, set a value between 0 and 127 - sizeof(my_data) to the offset variable, then set a non-zero value to the variables val_high and val_low, then verify that the my_data structure is equal to set_val.

```
u8 buffer[128]
i32 val_high
u32 val_low
stru_large_int set_val
stru_large_int my_data

; -------- loop begin --------
__MAIN_LOOP__

   ; range check
   if(offset >= 0 and (offset + sizeof(my_data)) < 128)
      set_val.high = val_high
      set_val.low  = val_low
      set_i64(buffer, offset, set_val)
      get_i64(buffer, offset, my_data)
   endif

END_MAIN_LOOP
```

© 2025 Robox SpA

Instruction get_u64

This function reads 8 bytes from a buffer, at the specified offset, interpreting them as a stru_large_uint structure and writing the value read into the given structure.

The mirror function is set_u64 747().

Syntax	**get_u64 (buffer, i32 offset, stru_large_uint data)**
buffer	Can be an array, structure, register
offset	Offset in bytes
data	Variable of type stru_large_uint 844 in which to save the read value
Validity	Rule, Task
Note	

Examples

Once you have loaded the example program on a control, via a shell you can verify the functioning of get_u64. To do this, set a value between 0 and 127 - sizeof(my_data) to the offset variable, then set a non-zero value to the variables val_high and val_low, then verify that the my_data structure is equal to set_val.

```
u8 buffer[128]
u32 val_high
u32 val_low
stru_large_uint set_val
stru_large_uint my_data

; -------- loop begin --------
__MAIN_LOOP__

   ; range check
   if(offset >= 0 and (offset + sizeof(my_data)) < 128)
      set_val.high = val_high
      set_val.low  = val_low
      set_u64(buffer, offset, set_val)
      get_u64(buffer, offset, my_data)
   endif

END_MAIN_LOOP
```

Instruction goto

This instruction transfers program execution to the desired instruction.

Syntax	**goto string label**
label	Label that identifies the program instruction from which to continue execution

© 2025 Robox SpA

Syntax	**goto string label**
Validity	Rule, Task
Note	You cannot jump outside of a function or rule

Labels can be a maximum of 32 characters long, and the first character must be alphabetic.

Subsequent characters may be alphabetic or numeric. The use of the low segment "_" is also allowed.

Examples

Example of use:

```
; jump to the label "PIPPO"
goto PIPPO
....
; label "PIPPO", new execution point
; note the presence of the colon ":"
PIPPO:
....
```

Instruction group

To fully understand the meaning of this fundamental instruction of the R3 programming language, the following must be kept in mind:

- In the synchronous task ($RULE 770 file) the user has the ability to write a very large number of routines called RULEs (rules). Each RULE has a specific task, and the fact that one can write so many of them gives the programmer the opportunity to subdivide the various problems he will face
- The synchronous task can execute up to 32 RULE executors(RCs) simultaneously
- To associate the executors (RC) with the RULE number, the programmer must use the group instruction
- To specify the order in which the executors (RCs) are to be processed, the programmer must use the order 708 instruction (defaults in sequence from 1 to 32)
- The predefined variable RC(n) identifies the nth of 32 executors. Its content is the RULE number associated with the nth executor via the group instruction
- The predefined variable OC(n) identifies the processing order of the 32 executors (RC)

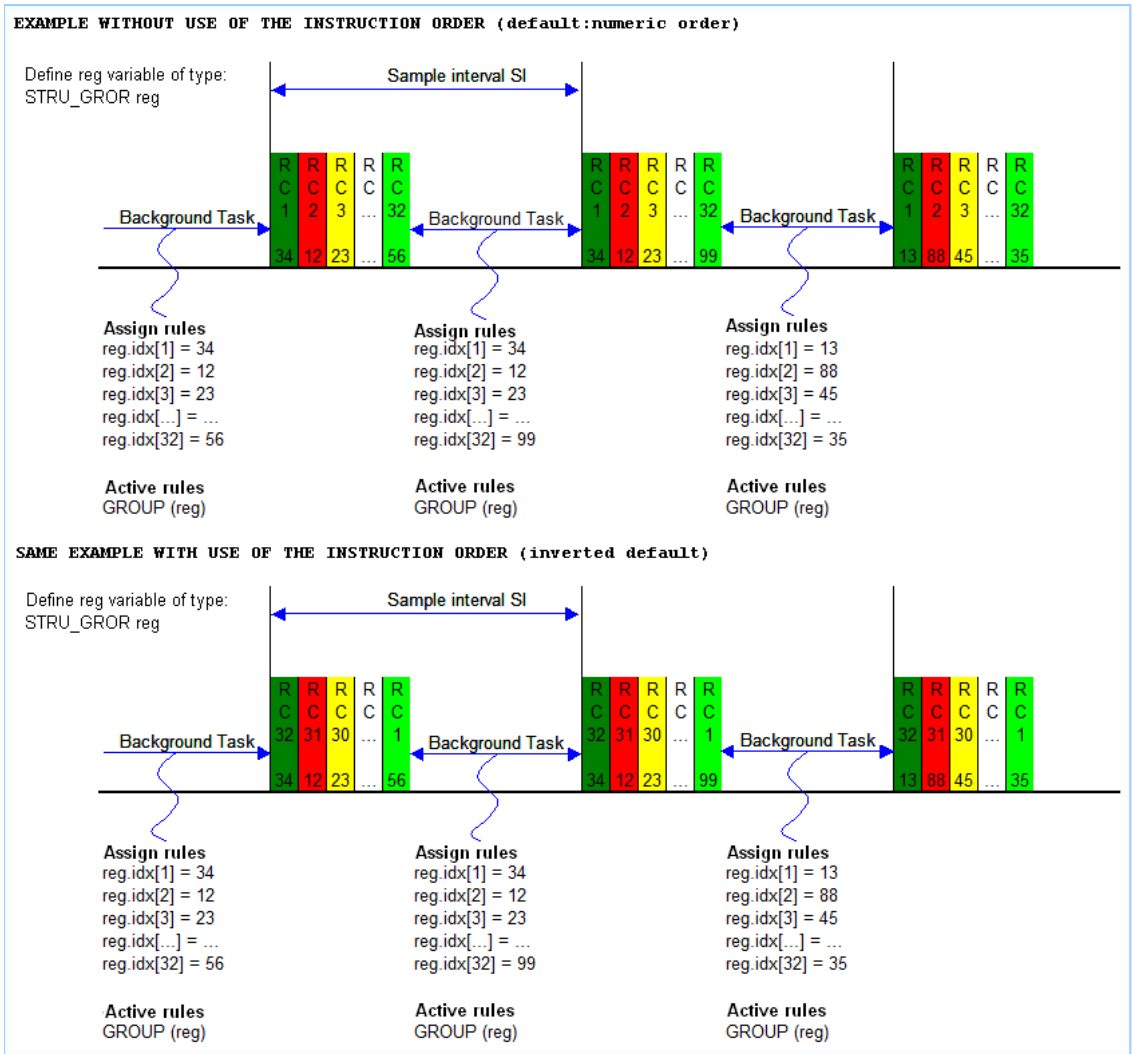
For better understanding, let us analyze the time when RTE is about to execute the synchronous task (the time interval is represented by the predefined variable si). RTE executes up to 32 RULE executors (RCs) which contain the RULEs triggered by the group instruction in the sequential order specified by the order instruction.

Syntax	**group stru_gror stru**
stru	Structure of type stru_gror 841 in which the rules will be inserted
Validity	Rule, Task
Note	

© 2025 Robox SpA

Examples

Example of use:



In the first example with default order we will have: OC(1) = 1 , OC(2) = 2 , ... , OC(32) = 32

In the first example with reversed order we will have: OC(1) = 32 , OC(31) = 2 , ... , OC(32) = 1

NOTE: If you want to change the rule of only one performer, you have to make sure that the structure remains congruous, as it is taken as a whole.

Example of use:

```
; variable definition assegna of type stru_gror
stru_gror 841 assegna
; rule executor rc(1) executes rule number 100
assegna.idx[1] = 100
; rule executor rc(2) executes rule number 200
assegna.idx[2] = 200
```

```
; rule executor rc(3) executes rule number 300
assegna.idx[3] = 300
; RTE will execute the 100,200,300 rules in the $RULE file
(synchronous task)
; In what order?
; RTE will respect the order imposed by the order instruction
group 694 (assegna)
```

Instruction if

This instruction allows the axis control that runs the R3 program to make logical decisions.

Syntax	**if (condition)** ;block1 [**else** ;block2] **endif**
Validity	Rule, Task
Note	

If the "condition" results TRUE, the instruction group defined as "block1" is executed.

If the "condition" results FALSE, the instruction group defined as "block2" is executed.

Examples

Examples of use:

```
; if the contents of the real register rr(2) is zero...
if(rr(2) = 0)
; I execute this instruction...
 rr(1) = rr(1) * 2
else 677
; otherwise I execute this other instruction...
 rr(1) = rr(1) / 2
; end of if-else-endif block
endif 683
```

Note: In case "block2" instructions do not need to be executed, the keyword else 677 can be omitted.

Example:

```
; LOOP label
LOOP:
; if the negate of START is TRUE...
if(NOT 782 (START))
; then jump to "LOOP"
 goto 693 LOOP
endif 683
```

```
; I get here only if START is true (i.e. non-zero)
```

Example:

```
; Nesting of if blocks is allowed:
; I check the status of UP
if( UP )
; if UP is TRUE I check if PIPPO is less than or equal to 255
    if( PIPPO <= 255 )
        PIPPO = PIPPO + 1
    endif 683
else 677
; if UP is FALSE I check if PIPPO is greater than or equal to 0
    if( PIPPO >= 0 )
        PIPPO = PIPPO - 1
    endif 683
endif 683
```

Instruction inf_report

This instruction allows a user-defined line of information to be written into the report, which can be inspected via the report command sent from the shell or via debugger from the project window.

Syntax	**inf_report (i32 numId, string info)**
numId	Allows a code to be set that, if in agreement with the predefined variable rep_mask, enables the message to be written to the report. **NOTE**: Valid values must be defined in the mask 0xFFE0. All other values will be switched to the value 0xFFFF00.
info	Line of information to be included in the report
Validity	Rule, Task
Note	Avoid recursive use of the function to not compromise execution timing

Examples

Example usage:

```
; I store the value of num_id in the variable "type"
type = 0x100
; I store the string of "info" in the string register sr(1)
sr(1) = "report test"
...
; if memo is TRUE, then I write the info line into the report
if 696 (memo)
```

© 2025 Robox SpA

```
 memo = 0
 inf_report(type, sr(1))
endif 683
```

Instruction int_after

Instruction that allows a rule to be sent to execution after a set time from its call. The rule must be written in an appropriate file beginning with the metacommand $RULE_TIME 770. This rule has higher priority than the synchronous rules 720, consequently the code written in it must be the minimum necessary!

Syntax	**int_after (i32 tempo, stru_int struct)**
tempo	Time after which to send the asynchronous rule into execution [1 - 65535 micro-seconds]
struct	Structure type stru_int 842 that contains related information (optional, if omitted disables instruction)
Validity	Rule, Task
Note	Instruction must be called only once. To change the time, the instruction must be called again. If the instruction is called with time value 0 or without the supporting data structure is disabled

Diagnostics

- 2802 Interrupt rule - unknown number <STRU_INT.nRule>
- 9900 ill.Arg A:1 T:x ST:xxx If the specified time is not in the allowed range [1 -:- 65535].
- 9900 ill.Arg A:2 T:x ST:xxx If the time rule task ($rule_time) has not been loaded into RAM (To load write command "load fa:mytask.cof" in autoexec.stp)
- 9901 ill.Funct. T:x ST:xxx If the timer interrupt has already been enabled.

Examples

Example of use:

You want to output a digital output for a specified time.

In the file mytask.r3:

```
$ rule_time
rule 720 100
 out(257) = 0
end_rule 682
```

In the main program (main program):

```
; I define the variable parAfter of type STRU_INT 842
```

© 2025 Robox SpA

```
stru_int parAfter
; execution one shot
parAfter.cntr.0 = 1
; The totalizer is used as a flag
parAfter.cntr.1 = 1
; rule present in the file mytask.r3
parAfter.nrule = 100
; we disable the axis quota photos
parAfter.maskPhoto = 0x0

...

LOOP:
; when you enable the input inp(2).
if 696 (rise(inp(2)))
 ; active digital output out(257)
 out(257) = 1
 ; enable time rule execution after 1 ms (it will reset my output
out(257))
 int_after(1000 , parAfter)
endif 683
...
goto 693 LOOP
```

Instruction int_inp

Instruction to send a rule into execution on the variation edge of a digital input that can generate interrupts. The rule must be written in an appropriate file beginning with the metacommand $RULE_INP 770. This rule has higher priority than the synchronous rules 720, consequently the code written in it must be the bare minimum!

Syntax	**int_inp (i32 numInp, stru_int struct [, i32 riseE=1] [, i32 fallE=0])**
numInp	Number of the input on which to observe the change
struct	Structure of type stru_int 842 that contains the related information (optional, if omitted disables the function)
riseE	Variation on the rising edge (optional, default 1)
fallE	Variation on the falling edge (optional, default 0)
Validity	Rule, Task
Note	Instruction must be called only once. If the instruction is called without the supporting data structure, it is disabled

© 2025 Robox SpA

Diagnostics

- 2802 Interrupt rule - unknown number <STRU_INT.nRule>.
- 9900 ill.Arg A:1 T:x ST:xxx If the indicated digital input is not interrupt, or if the requested edge is not programmable
- 9900 ill.Arg A:3 T:x ST:xxx If both r_e and f_e are null
- 9900 ill.Arg A:-1 T:x ST:xxx If illegal internal channel
- 9900 ill.Arg A:-2 T:x ST:xxx If channel already programmed, if the file containing the specified rules is not loaded into memory
- 9900 ill.Arg A:-6 T:x ST:xxx Error during interrupt programming (e.g., specified edge not managed)

Examples

Example of use:

```
; We want to capture the real quota cp on the change in input 257.
; In that case the information that the predefined variable
STRU_INT 842 gives us.
; is sufficient so we can avoid writing the asynchronous rule.
; In the main program (task1.r3):
; I define the variable parInp of type STRU_INT 842
STRU_INT 842 parInp
; continuous execution
parInp.cntr.0 = 0
; The totalizer is used as the flag
parInp.cntr.1 = 1
; No asynchronous rule (it is not mandatory to write the rule)
parInp.nrule = 0
; we enable photo on real quota of the first 4 axes
parInp.maskPhoto = 0xF
...
; start code
; enable photo on input variation
int_inp(257 , parInp)
LOOP:
; if photo has been executed
if 696 (rise(parInp .count = 1))
; I reset the flag of executed photo
parInp .count = 0
; we support photos on input on support variables
appoQuotaAx_1 = parInp.pos [1]
appoQuotaAx_2 = parInp.pos [2]
appoQuotaAx_3 = parInp.pos [3]
appoQuotaAx_4 = parInp.pos [4]
endif 683
...
goto 693 LOOP
; If on the variation we had wanted to send a rule into execution.
```

© 2025 Robox SpA

```
; we would have proceeded as follows:
; Initialize (from task1.r3) the variable parInp.nrule to the value
456
parInp.nrule = 456
```

Creation of the asynchronous rule (task2.r3):

```
$rule_inp
rule 720 456
 aux 720
 ; we increment the register rr(1))
 inc 351 (rr(1))
 end_aux 678
end_rule 682
```

Instruction int_timer

Instruction that allows a rule to be sent to execution at the set frequency. The rule must be written in an appropriate file beginning with the metacommand $RULE_TIME 770. This rule has higher priority than the synchronous rules 720 consequently the code written in it must be the minimum necessary!

Syntax	**int_timer (i32 freq, stru_int struct)**
freq	Execution frequency of the asynchronous timed rule [16 - 5000 Hz]
struct	Structure of type stru_int 842 containing related information (optional, if omitted disables function)
Validity	Rule, Task
Note	Instruction must be called only once. Instruction must be called again to change the frequency. If the instruction is called with frequency value 0 or without the supporting data structure is disabled

Diagnostics

- 2802 Interrupt rule - unknown number <STRU_INT.nRule>
- 9900 ill.Arg A:1 T:x ST:xxx If the frequency is not in the allowed range [16 -:- 5000 Hz].
- 9900 ill.Arg A:2 T:x ST:xxx If the time rule task ($rule_time) has not been loaded into RAM.
- 9901 ill.Funct. T:x ST:xxx If the timer interrupt has already been enabled.

Examples

Example usage:

In the file mytask.r3 :

© 2025 Robox SpA

```
$ rule_time
rule 100
      AUX
            out(257) = not out(257)
      END_AUX
end_rule
```

In the main program:

```
; I define the parTimer variable of type STRU_INT
STRU_INT parTimer
; continuous execution
parTimer.cntr.0 = 0
; The totalizer is enabled to count
parTimer.cntr.1 = 0
; rule present in the file mytask.r3
parTimer.nrule = 100
; we disable real quota photos
parTimer.maskPhoto = 0x0
...
LOOP:
if(enable)
 ; enable time rule execution at a frequency of 2KHz
 ; the square wave will then have frequency 1KHz
 int_timer(2000 , parTimer)
else
 ; I disable timed rule execution at the frequency of 2KHz
 int_timer(0)
endif
goto LOOP
```

Instruction lit

Defines a text string with which a symbolic name is associated. This name can be used repeatedly within the R3 program and is automatically recognized at compile time.

Syntax	**lit string name string str**
name	It is the symbolic name associated with the text string. It must adhere to the following rules: • Maximum 60 characters • The first character must be a letter
str	It is the text string that replaces the symbolic name whenever it is encountered. The limitations are as follows: • Maximum string length 190 characters
Validity	Rule, Task

© 2025 Robox SpA

Syntax	**lit string name string str**
Note	Symbolic names are recognized indifferently if they appear in lower or upper case. A name is recognized only if it forms a word by itself, while it is ignored if it is part of a longer word. Symbolic names are not allowed to be defined equal to any reserved word of the R3 language

The purpose of this command is to return either a more compact or more intelligible program source.

The definition must precede use; otherwise a compiler error message will result.

Instruction literally does not cause a lengthening of the object code produced by the compiler; it may, however, cause a lengthening of compile time.

Examples

Example of file lithium.i3:

```
lit iAirCompressed inp(257)
lit oSiren out(257)
lit rrSpeedManual nvrr(1)
lit rrAccAuto nvrr(123)
```

Example use in code:

```
; I check compressed air supply status and activate the siren if it
lacks pressure
if(not iAirCompressed)
 oSiren = true
else
 oSiren = false
endif

; I define the values of the machine's retentive registers
rrSpeedManual = 12.5
rrAccAuto = 10000.0
```

Instruction motion

Start of motion block within a rule.

Syntax	**motion**
Validity	Rule
Note	

Field motion - end_motion 681 contains the code for generating the ideal trajectory.

Examples

Example of generic rule:

© 2025 Robox SpA

```
; beginning of Rule
rule 720 1
 ...
 ; block ref (optional)
 ref 717
 ...
 ; end block ref
 end_ref 681
 ; motion block (required)
 motion
 ...
 iv(1) = 0
 ...
 ; end motion block
 end_motion 681
 ; aux block (optional)
 aux 652
 ...
 ; end aux block
 end_aux 678
 ...
 ; end of rule
end_rule 682
```

Instruction move_byte

This instruction copies the specified number of bytes from the source address to the destination address.

Syntax	**move_byte (sorg, dest, i32 num)**
sorg	It is the address at which the memory area to be copied begins. It can be an integer, real or string local variable or an integer or real or string register
dest	It is the address at which the memory area where to download data begins. It can be an integer, real, or string local variable or an integer or real or string register
num	It is the number of bytes to be copied
Validity	Rule, Task
Note	The system does not guarantee atomicity of the buffer. Special care must be taken if the data moved may be changed by multiple tasks with different priorities

Examples

Example of use:

© 2025 Robox SpA

```
; I copy the registers r(1) to r(1001) for a length of 100 bytes
; since each register occupies 4 bytes, 25 registers will be copied
move_byte(r(1), r(1001), 100)
```

Instruction move_mem

This instruction allows a portion of memory of specified length to be copied from a source location to a destination location.

Syntax	**move_mem (sorg, dest, num)**
sorg	It is the variable from which you intend to start copying. It cannot be a mathematical expression. It can be an integer, real or string register, or a local integer, real or string variable
dest	It is the variable from whose location the recopied block of memory is downloaded. It cannot be a mathematical expression. It can be an integer, real or string register, or an integer, real or string local variable. In the case of registers, the copying is extended to the registers following the one specified as the destination. In the case of local variables, which will typically have at least one index, care must be taken not to attempt to transfer more than the declared size of the destination variable. Failure to do so will result in an appropriate error message during program execution. This check is performed only in the case where *dest* is a local variable
num	It is the number of elements to be copied. It can be a mathematical expression. The number *num* does not represent a number of bytes, but a number of elements; e.g., in the case of real variables, if you specify *num* = 5, five consecutive real locations of the specified variable will be copied (in total, 40 bytes). The type of the variable will be the same as the type of *sorg*
Validity	Rule, Task
Note	The system does not guarantee atomicity of the buffer. Special care must be taken if the data moved may be changed by multiple tasks at different priorities

In the case of strings, the character 0 (string terminator) is copied only if it is included in the number of characters whose copying is requested. Consequently, to transfer the string "hello" entirely, a move_mem with *num* = 5 must be performed.

Examples

© 2025 Robox SpA

Example of use:

```
; definition of variables
real 898 pluto[40]
string 899 message
; the 20 real registers from register rr(5) to register rr(24)
; are copied into the one-dimensional array pluto
move_mem(rr(5), pluto[0], 20)
; the first 40 characters of the string register sr(3)
; are copied into the local message string
move_mem(sr(3), message, 40)
```

Instruction mt_dis

Allows disabling RTE execution of the specified time-sharing task(s).

Syntax	**mt_dis (n, n, n....)**
n	Number of the task to be disabled
Validity	Task
Note	Remember that after the boot phase rte sends only task 1 into execution. It is up to the user application, to invoke the rules 720 and enable/disable or put on hold the remaining tasks

Examples

Example of use:

```
; disables execution by RTE of task 2
mt_dis(2)
; disables execution by RTE of all tasks excluding task 1
mt_dis(2,3,4,5,6,7,8)
```

Instruction mt_en

Enables RTE execution of the specified time-sharing task(s).

Syntax	**mt_en (n, n, n....)**
n	Number of the task to be enabled
Validity	Task
Note	Remember that after the boot phase rte sends only task 1 into execution. It is up to the 'user application, to invoke the rules 720 and enable/disable or put on hold the remaining tasks.

© 2025 Robox SpA

Syntax	**mt_en (n, n, n....)**
	The command to load the task from flash-card to ram (e.g., LOAD fa:taskname.cof) must be present

Examples

Example of use:

```
; enables the execution of task 2
mt_en(2)
; enables execution of all 8 tasks
mt_en(2,3,4,5,6,7,8)
```

Instruction mt_rr_flag

NOTE: Deprecated.

This instruction allows you to be able to send an infinite OB into execution (loop).

Setting BIT0 RTE to 1 will take care of running the OB in partition of time along with the other tasks.

The same thing could be achieved previously by acting on the OB source via the tswait instruction.

Syntax	**mt_rr_flag = value**
bit0	Disable step-synchronous task switching. To be set before a long OB FUNCTION CALL or always. To be restore to 0 immediately after the OB FUNCTION CALL
Validity	Task
Note	

Instruction mt_wait

This instruction stops the program (thus the task) until the condition expressed becomes true.

Syntax	**mt_wait (cond)**
cond	It is any expression
Validity	Task
Note	It should normally be used in tasks whose execution is required on command (event). In fact, tasks, by standing idle, leave more execution time for other tasks (time-sharing).

© 2025 Robox SpA

Syntax	**mt_wait (cond)**
	To stop the task for an interval of time use dwell 677

Examples

Examples of use:

```
; The task remains stationary until one of two conditions occurs.
mt_wait(inp(3) or out(17))
; The task remains stopped until the volatile integer register r(17)
takes value 1000
mt_wait(r(17) = 1000)

; Passing constant 0 stops the task forever
mt_wait(0)

;If you pass a constant other than 0, even decimal, the function
never blocks
mt_wait(0.2)
```

Instruction order

This instruction allows defining the order in which RTE will process the rule executors (rc) activated by the group instruction.

Syntax	**order stru_gror struct**
struct	Structure of type stru_gror 841
Validity	Rule, Task
Note	By default they are all enabled and follow the natural order. (rc(1)=first , rc(2)=second ,..., rc(32)=last) **NOTE**: To disable RTE from executing rc's, simply assign the value -1 to the executor.

Examples

Examples of use:

```
STRU_GROR ordine
STRU_GROR assegna
; we establish the execution sequence of the rule executors (rc).
; executor 3 (rc(3)) will be executed first (=>RULE #30)
ordine.idx[1] = 3
; executor 2 (rc(2)) will be executed second (=>RULE #20)
ordine.idx[2] = 2
; executor 1 (rc(1)) will be executed third (=>RULE #10)
```

© 2025 Robox SpA

```
ordine.idx[3] = 1
; from the fourth onward they are disabled.
ordine.idx[4] = -1
; I execute the operation that defines the execution order
order(ordine)
; we determine which rule executors are to execute.
; the rule executor rc(1) executes RULE number 10
assegna.idx[1] = 10
; the rule executor rc(2) executes RULE number 20
assegna.idx[2] = 20
; the rule executor rc(3) executes RULE number 30
assegna.idx[3] = 30
; I perform the operation that assigns the rules to the executors.
group(assegna)
; RTE will execute in order the rules 30, 20, 10 found in the $RULE
file (fixed frequency tasks).
; To get an estimate of the time duration of this execution use the
default variable rule_lenght
```

```
; definition of the variable assignRules of type STRU_GROR.
STRU_GROR assegnaRegole
; code
; the rule executor rc(1) executes RULE number 136
assegnaRegole.idx[1] = 136
; the rule executor rc(2) executes RULE number 245
assegnaRegole.idx[2] = 245
; the rule executor rc(3) executes RULE number 36
assegnaRegole.idx[3] = 36
; I perform the operation that assigns the rules to the executors.
group(assegnaRegole)
; RTE will execute rules 136, 245, 36 in the $RULE file (fixed-
frequency tasks).
; In what order?
; RTE will respect the order set by the order instruction.
```

```
; definition of order variables and assignRules of type STRU_GROR.
STRU_GROR ordine
STRU_GROR assegnaRegole
; code
; we establish the execution sequence of the rule executors (rc).
; executor 3 (rc(3)) will be executed first (=>RULE #30)
ordine.idx[1] = 3
; executor 2 (rc(2)) will be executed second (=>RULE #20)
ordine.idx[2] = 2
; executor 1 (rc(1)) will be executed third (=>RULE #10)
ordine.idx[3] = 1
; executors 4 to 32 are disabled - The result is a gain in machine
time!
ordine.idx[4] = -1
; I execute the operation that defines the execution order
```

© 2025 Robox SpA

```
order(ordine)
; the rule executor rc(1) executes RULE number 136
assegnaRegole.idx[1] = 136
; the rule executor rc(2) executes RULE number 245
assegnaRegole.idx[2] = 245
; the rule executor rc(3) executes RULE number 36
assegnaRegole.idx[3] = 36
; I perform the operation that assigns the rules to the executors.
group(assegnaRegole)
; RTE will execute rules 136, 245, 36 in the $RULE file (fixed-
frequency tasks).
; In what order?
; RTE will respect the order set by the order instruction.
```

Instruction out_m

This instruction is used to change Boolean variables whether they are logical or physical. It sets the state of the variable permanently until a subsequent similar instruction changes its state further.

Syntax	**out_m (s ch, s ch, ..., s ch)**
s	Represents the state of the boolean variable: • - = FALSE (open if physical output) • + = TRUE (closed if physical output) (can be omitted)
ch	This is the index that identifies the OUTPUT variable. It can be an integer expression and can range from 1 to 2304
Validity	Rule, Task
Note	You can do the same operation directly by imposing the state of the physical channel. ex: OUT(n) = 1 or OUT(n) = 0

Up to 10 outputs may be specified in an instruction (compile-time error reporting if exceeded). Indexes must be separated from each other by the comma (,) character.

Diagnostics

- 9900 ill. Arg A:1 T:x ST:xxx if the number of managed out is greater than 10 (where x = number of tasks and xxx number of steps)

Examples

Example of use:

```
; the output of channel 12 is activated
```

© 2025 Robox SpA

```
out_m(12)
; the output of channel 12 is turned off
out_m(-12)

; the output of channel 56 is turned on
; while the output of channel 24 is deactivated
out_m(+56 , -24)
; if(excite) is turned on...
if 696 (excite)
; the outputs of channels 257 to 266 are activated
 out_m(257,258,259,260,261,262,263,264,265,266)
else 677
; else the outputs of channels 257 to 266 are disabled
 out_m(-257,-258,-259,-260,-261,-262,-263,-264,-265,-266)
endif 683
```

Instruction qwbe2le

Converts big endian qword (x64) to little endian qword (x64).

Syntax	**qwbe2le (stru_large_int|stru_large_uint stru)**
stru	Structure type stru_large_int 844 or stru_large_uint 844 to convert
Validity	Task, Rule
Note	

Examples

Example of use:

```
STRU_LARGE_UINT input

input.low  = 0x01020304
input.high = 0x05060708
QWBE2LE (input)
; input.low = 0x08070605, input.high = 0x04030201
```

Instruction qwbedwx2le

Converts dword exchange big endian qword (x64) to little endian qword (x64).

Syntax	**qwbedwx2le (stru_large_int| stru_large_uint stru)**
stru	Structure of type stru_large_int 844 or stru_large_uint 844 to be converted

© 2025 Robox SpA

Syntax	**qwbedwx2le (stru_large_int| stru_large_uint stru)**
Validity	Task, Rule
Note	

Esempi

Example of use:

```
STRU_LARGE_UINT input

input.low  = 0x05060708
input.high = 0x01020304
QWBEDWX2LE (input)
; input.low = 0x08070605, input.high = 0x04030201
```

Instruction qwle2be

Converts little endian qword (x64) to big endian qword (x64).

Syntax	**qwle2be (stru_large_int|stru_large_uint stru)**
stru	Structure type stru_large_int 844 or stru_large_uint 844 to be converted
Validity	Task, Rule
Note	

Examples

Example of use:

```
STRU_LARGE_UINT input

input.low  = 0x08070605
input.high = 0x04030201
QWLE2BE (input)
; input.low = 0x01020304, input.high = 0x05060708
```

Instruction qwle2bedwx

Converts little endian qword (x64) to dword exchange big endian qword (x64).

Syntax	**qwle2bedwx (stru_large_int| stru_large_uint stru)**
stru	Structure type stru_large_int 844 or stru_large_uint 844 to convert

© 2025 Robox SpA

Syntax	**qwle2bedwx (stru_large_int| stru_large_uint stru)**
Validity	Task, Rule
Note	

Examples

Example of use:

```
STRU_LARGE_UINT input

input.low  = 0x08070605
input.high = 0x04030201
QWLE2BEDWX (input)
; input.low = 0x05060708, input.high = 0x01020304
```

Instruction qwle2bedwxf

Converts little endian qword (x64) to dword exchange big endian qword (x64).

Syntax	**qwle2bedwxf (real data, stru_large_int| stru_large_uint stru)**
data	Data input
stru	Structure of type stru_large_int 844 or stru_large_uint 844 to be converted
Validity	Task, Rule
Note	

Examples

Example of use:

```
REAL input
STRU_LARGE_UINT result

input = 3.141592653589793116 ; Data memory: 18 2d 44 54 21 09 40 hex.
QWLE2BEDWXF(input, result)
; result.low  = 0x182d4454 -> data memory: 54 44 2d 18 hex.
; result.high = 0xfb210940 -> data memory: 40 09 21 fb hex.
```

© 2025 Robox SpA

Instruction qwle2bef

Converts little endian real (x64) to big endian qword (x64).

Syntax	**qwle2bef (real data, stru_large_int| stru_large_uint stru)**
data	Data input
stru	Structure type stru_large_int 844 or stru_large_uint 844 to convert
Validity	Task, Rule
Note	

Examples

Example of use:

```
REAL input
STRU_LARGE_UINT result

input = 3.141592653589793116 ; Data memory: 18 2d 44 54 21 09 40 hex.
QWLE2BEF (input, result)
; result.low  = 0xfb210940 -> data memory: 40 09 21 fb hex.
; result.high = 0x182d4454 -> data memory: 54 44 2d 18 hex.
```

Instruction qwle2mbe

Converts little endian qword (x64) to middle big endian qword (x64).

Syntax	**qwle2mbe (stru_large_int|stru_large_uint stru)**
stru	Structure type stru_large_int 844 or stru_large_uint 844 to be converted
Validity	Task, Rule
Note	

Examples

Example of use:

```
STRU_LARGE_UINT input

input.low  = 0x08070605
input.high = 0x04030201
QWLE2MBE (input)
; input.low = 0x03040102, input.high = 0x07080506
```

© 2025 Robox SpA

Instruction qwle2mbedwx

Converts little endian qword (x64) to dword exchange middle big endian qword (x64).

Syntax	**qwle2mbedwx (stru_large_int| stru_large_uint stru)**
stru	Structure type stru_large_int 844 or stru_large_uint 844 to be converted
Validity	Task, Rule
Note	

Examples

Example of use:

```
STRU_LARGE_UINT input

input.low  = 0x08070605
input.high = 0x04030201
QWLE2MBEDWX (input)
; input.low = 0x07080506, input.high = 0x03040102
```

Instruction qwle2mbedwxf

Converts little endian real (x64) to dword exchange middle big endian qword (x64).

Syntax	**qwle2mbedwxf (real data, stru_large_int| stru_large_uint stru)**
data	Data input
stru	Structure type stru_large_int 844 or stru_large_uint 844 to convert
Validity	Task, Rule
Note	

Examples

Example of use:

```
REAL input
STRU_LARGE_UINT result

input = 3.141592653589793116 ; Data memory: 18 2d 44 54 21 09 40 hex.
QWLE2MBEDWXF (input, result)
; result.low  = 0x4454182d -> data memory: 2d 18 54 44 hex.
; result.high = 0x0940fb21 -> data memory: 21 fb 40 09 hex.
```

© 2025 Robox SpA

Instruction qwle2mbef

Converts little endian real (x64) to middle big endian qword (x64).

Syntax	**qwle2mbef (real data, stru_large_int| stru_large_uint stru)**
data	Data input
stru	Structure type stru_large_int 844 or stru_large_uint 844 to convert
Validity	Task, Rule
Note	

Examples

Example of use:

```
REAL input
STRU_LARGE_UINT result

input = 3.141592653589793116 ; Data memory: 18 2d 44 54 21 09 40 hex.
QWLE2MBEF (input, result)
; result.low  = 0x0940fb21 -> data memory: 21 fb 40 09 hex.
; result.high = 0x4454282d -> data memory: 2d 18 54 44 hex.
```

Instruction qwmbe2le

Converts middle big endian qword (x64) to little endian qword (x64).

Syntax	**qwmbe2le (stru_large_int|stru_large_uint stru)**
stru	Structure type stru_large_int 844 or stru_large_uint 844 to convert
Validity	Task, Rule
Note	

Examples

Example of use:

```
STRU_LARGE_UINT input

input.low  = 0x03040102
input.high = 0x07080506
QWMBE2LE (input)
; input.low = 0x08070605, input.high = 0x04030201
```

© 2025 Robox SpA

Instruction qwmbedwx2le

Converts dword exchange middle big endian qword (x64) to little endian qword (x64).

Syntax	**qwmbedwx2le (stru_large_int| stru_large_uint stru)**
stru	Structure type stru_large_int 844 or stru_large_uint 844 to convert
Validity	Task, Rule
Note	

Examples

Example of use:

```
STRU_LARGE_UINT input

input.low  = 0x07080506
input.high = 0x03040102
QWMBEDWX2LE (input)
; input.low = 0x08070605, input.high = 0x04030201
```

Instruction ref

Start of ref block within a rule 720.

Syntax	**ref**
Validity	Rule
Note	

If present, the REF field must be declared as the first field within the Rule. Failure to do so results in an error being reported in compilation.

The REF field of each RULE typically sets, via the sref variable, the value that will be sent as a reference to the drives of the axis(s) involved (use tref in case of speed loop closure on axis control).

If no value is assigned to the axis sref in the REF field, the value used as a reference will be that of the previous sampling (similar behavior will occur in case of tref assignment).

The value assigned to the sref variables can be the result of any expression: it will still be made integer and limited within the range of action of the reference emission device configured in the RTE.CFG file (similar behavior will occur in case of tref assignment).

If the sr_ovf_m alarm is enabled for the axis there will be the output of that alarm whenever, at runtime, the value assigned to the sref variable is outside the variation range mentioned above (similar behavior will occur in the case of tref assignment and tr_ovf_m alarm enabled).

Any assignment of a value to the sref or tref variables outside the REF field is considered an error by the compiler.

NOTE: May be omitted.

© 2025 Robox SpA

Examples

Example of generic rule with REF block:

```
; beginning of Rule
rule 720 1
 ...
 ; block ref (optional)
 ref
 ...
 sref(1) = ...
 ...
 ; end block ref
 end_ref 681
 ; motion block (required)
 motion 703
 ...
 ; end motion block
 end_motion 681
 ; aux block (optional)
 aux 652
 ...
 ; end aux block
 end_aux 678
 ...
; end of rule
end_rule 682
```

Instruction ref_prologue

With this instruction, it is possible to specify the name of a function that will be executed at each sampling of the rules 720 (which run at a fixed frequency set via the rule_freq 729 instruction) before the ref 717 fields are executed.

Syntax	**ref_prologue functionName**
functionName	It is the name of the function to be sent to execution (without parameters and without return value)
Validity	Rule
Note	The instruction must be called only once. If the instruction is called by omitting *functionName* (= 0) it is disabled

Examples

Example of use:

```
function 688 rule_init 730()
ref_prologue 718(refPrologue)
```

```
    ; ...
end_fun

function refPrologue
    ; ...
end_fun
```

Instruction restart_point

This instruction imposes an interruption of task execution on a live modification request.

Syntax	**restart_point**
Validity	Task
Note	Can be more than one

Examples

Example of use:

```
; I call an initialization routine
if(rFirstBoot)
call 667 initParam
endif
; start of main loop
__main_loop__
; main loop instruction
restart_point ...
 ...
; I measure the duration of the main loop and store the value in the
variable T1
 T1 = loop_time 363()
; I run statistics on the duration of the main loop
 returns = statistics 595 (T1,statT1)
; end of main loop --> return to beginning
end_main_loop 680
```

Instruction return

Interruption of the function invoked by the call 667 instruction with possible return of value.

Syntax	**return val**
val	Value that is returned at the location of the function call (optional)
Validity	Rule, Task
Note	The type of the return value depends on the declaration of the function

© 2025 Robox SpA

Examples

Example of use:

```
; function call without using the return value
call 667 calc_diag1(12,17)
; function call inside a mathematical expression
; inside the expression the function call is equivalent to a variable
of type REAL
rr(1) = 5.0 + calc_diag2(12,17)
; function calc_diag1
function 688 calc_diag1 (REAL val1, REAL val2)
    if 696 ((val1<0) or (val2<0))
        rr(1) = 0
        return
    endif 683
    rr(1) = sqrt 594(val1*val1 + val2*val2)
end_fun 680
; function calc_diag2
function 688 REAL 898 calc_diag2 (REAL val1, REAL val2)
    if 696 ((val1<0) or (val2<0))
        return 0.0
    endif 683
    return sqrt(val1*val1 + val2*val2)
end_fun 680
```

Instruction rule

Identifier of a block of instructions (executed on interrupt($RULE 770) at a frequency set by the rule_freq instruction) intended to describe and determine the rules of behavior of one or more axes.

Syntax	**RULE num** **axes** n[,n,n,...,n] | **axes_m** m **ref** ; REF BLOCK (optional) **end_ref** **motion** **if**(first_time()) ; only the first time **endif** ; MOTION LOCK **end_motion** **aux** ; AUX BLOCK (optional) **end_aux** **END_RULE**
num	It is the identification number of the rule.

Syntax	**RULE num** **axes** n[,n,n,...,n] | **axes_m** m **ref** ; REF BLOCK (optional) **end_ref** **motion** **if**(first_time()) ; only the first time **endif** ; MOTION LOCK **end_motion** **aux** ; AUX BLOCK (optional) **end_aux** **END_RULE**
	It must be a constant. Within the $RULE 770 file the user can write infinite rules. Up to 32 rule executors(rc) can be active simultaneously in the sampling. To decide which rules to run in the next sampling, the user has the group 694 instruction, and to decide the order of execution in the sampling, the user has the order 708 instruction. • order: instruction used only once in initialization • group: widely used instruction since the active rules must change according to the state of the machine
axes (n[,n,n])	(alternative to *axes_m*) This is the identifier used to declare which axes are manipulated by the rule (1-:-32). *n* are the axis identifiers. At least one axis must be declared
axes_m (mask)	(alternative to *axes*) Allows the mask of the axes involved to be written (the value can be an expression). **Note**: This declaration is also optional. No checks are performed in either compilation or execution.
ref	If omitted, the compiler will handle the position loop closure algorithm by the following default formula "Proportional position loop closure with velocity feed forward." epos(n) = p_ip(n, ipp_idx) - cp(n) sref(n) = epos(n) * pro_gai(n) + kff(n) * iv(n) The user will simply have to use the predefined variables pro_gai and kff to calibrate the axes. If you do not wish to have a speed feed forward component, simply set the kff variables of the affected

© 2025 Robox SpA

Syntax	**RULE num** **axes** n[,n,n,...,n] | **axes_m** m **ref** ; REF BLOCK (optional) **end_ref** **motion** **if**(first_time()) ; only the first time **endif** ; MOTION LOCK **end_motion** **aux** ; AUX BLOCK (optional) **end_aux** **END_RULE**
	axes to 0. If, on the other hand, a null value of pro_gai is imposed, open-loop axis drive is achieved. Keep in mind that with purely proportional feedback, the position error that is obtained is directly proportional to the velocity. The feed forward component is inserted to bring the error in velocity to zero. The user will need to set the values of ser_thr and ser_gai appropriately for proper tracking alarm(sam) handling. Remember that in the predefined variable ser_cthr RTE provides the current threshold (calculated sampling by sampling) of theservo alarm. Use the oscilloscope tool from RDE to set the correct ser_thr and ser_gai values
end_ref	End of block ref
motion	The rule motion field describes the equations that govern the state of the system governed by Robox axis controls and programmed in R3 language. The equations will bind the state variables of the system, variables that can be, in general, of any type, but, given the type of Robox steering units, they will be fundamentally kinematic, namely, positions, velocities, accelerations. That is, in R3 language: • ip (ideal position) i.e. • iv (ideal velocity) i.e. the ideal speed • ia (ideal acceleration) i.e., the ideal acceleration and also: • cp (current position) i.e. the actual position • cv (current velocity) i.e., the actual velocity • ca (current acceleration) i.e., the actual acceleration Where ideal units are the reference units, that is, the value that, at a given time, one would like position,

© 2025 Robox SpA

Syntax	**RULE num** **axes** n[,n,n,...,n] | **axes_m** m **ref** ; REF BLOCK (optional) **end_ref** **motion** **if**(first_time()) ; only the first time **endif** ; MOTION LOCK **end_motion** **aux** ; AUX BLOCK (optional) **end_aux** **END_RULE**
	velocity and acceleration to have. The actual quantities are the actual values of position, velocity and acceleration as measured through the appropriate transducers. The measured quantity is usually position, so actual velocity and acceleration are derived by numerical derivation. Velocity will be the change in position over a sampling interval (equal to si); similarly, acceleration will be the change in velocity over a sampling interval. It should be focused that the ideal position (ip) represents the desired position for a certain axis at a certain time. It should not be confused with the position where you want the axis to go, obeying a certain law of motion (and therefore taking a certain amount of time). In the motion field of each rule, the law of motion of one or more axes is expressed. One and only one of the variables ip, iv, ia may be represented there in explicit form (i.e., appear in an assignment instruction to the left of the "=" sign). It is not permissible to assign values to more than one of the above three variables. RTE takes charge of correctly calculating the two missing variables in the manner given below. If two or three variables, referring to the same axis, are assigned in the motion field, RTE generatesalarm 2800 during execution. If none of the three variables are assigned explicitly, there is no error reporting by the compiler, but some important checks(servo alarm, crash alarm, speed alarm etc.) are missed during execution; the automatic calculation of the remaining axis-related variables is also missed. Multiple axis-related laws are allowed in the RULE. It is inadvisable to use the same RULE to describe the motion of mutually independent axes. In particular:

© 2025 Robox SpA

Syntax	**RULE num** **axes** n[,n,n,...,n] | **axes_m** m **ref** ; REF BLOCK (optional) **end_ref** **motion** **if**(first_time()) ; only the first time **endif** ; MOTION LOCK **end_motion** **aux** ; AUX BLOCK (optional) **end_aux** **END_RULE**
	• IF IP IS ASSIGNED THE BEHAVIOR WILL BE AS FOLLOWS: If the set position is greater than/minor than the software limit switches parametrized in the predefined variables max_str(n) and min_str(n) and the predefined variable cam is enabled then: * If the ideal quota ip is outside the software limit switches max_str and min_str and you want to move further away RTE generates the crash alarm and the imposed IP is ignored (the previous one is kept). * If the ideal quota ip is on the software endpoints max_str and min_str and you want to move out RTE generates the crash alarm and the imposed ip is limited to the relative end of strokes. As long as the ip is outside the end of strokes, the alarm cannot be reset. By acting on the predefined variable ip_write_bounds, changing its default value (0xFFFFFF), it will be possible to change the behavior as follows: * If the ideal ip quota is on the software limit switch max_str and min_str and you want to exit RTE generates the crash alarm and the imposed ip, even if outside the end of strokes is assumed. Even if the ip results outside the end of strokes it will be possible to reset the alarm. It will be regenerated if an attempt is made to move further away. The speed is then calculated by RTE. If it results greater than the maximum speed parametrized in the predefined variables max_spe(n) and the spam alarm is enabled, the speed alarm is issued. The acceleration is calculated by RTE. If it results greater than the maximum acceleration parametrized in the predefined variables max_acc(n) and the acam alarm is enabled, the acceleration alarm is issued.

© 2025 Robox SpA

Syntax	**RULE num** **axes** n[,n,n,...,n] | **axes_m** m **ref** ; REF BLOCK (optional) **end_ref** **motion** **if**(first_time()) ; only the first time **endif** ; MOTION LOCK **end_motion** **aux** ; AUX BLOCK (optional) **end_aux** **END_RULE**
	The Servo (tracking) alarm is handled if enabled via the predefined variable sam
	• IF IV IS ASSIGNED, THE BEHAVIOR WILL BE AS FOLLOWS: If the set speed is greater than the maximum speed parametrized in the predefined variables max_spe(n): * It is limited to the maximum speed itself (max_spe(n)) if and only if the predefined variable spam is enabled. * A warning is given to the user via the predefined variable splm if and only if the spam predefined variable is enabled. The acceleration is calculated by RTE. If it results greater than the maximum acceleration parametrized in the predefined variable max_acc(n): * Acceleration alarm is generated if and only if the predefined variable acam is enabled. The position is then calculated by RTE based on the formula: ip = ip + iv * si If the position results greater than/minor than the software limit switches parametrized in the predefined variables max_str(n) and min_str(n) and the predefined variable cam is enabled, the 'crash alarm (lower and upper) is issued. The Servo (tracking) alarm is handled if enabled via the predefined variable sam
	• SE È ASSEGNATO IA IL COMPORTAMENTO SARÀ IL SEGUENTE: Se l'accelerazione è maggiore della accelerazione massima parametrizzata nelle variabili predefinite max_acc(n): * Viene limitata alla accelerazione massima stessa se e solo se la variabile predefinita acam è abilitata.

© 2025 Robox SpA

Syntax	**RULE num** **axes** n[,n,n,...,n] | **axes_m** m **ref** ; REF BLOCK (optional) **end_ref** **motion** **if**(first_time()) ; only the first time **endif** ; MOTION LOCK **end_motion** **aux** ; AUX BLOCK (optional) **end_aux** **END_RULE**
	* Viene data segnalazione all'utente tramite la variabile predefinita acclm se e solo se la variabile predefinita acam è abilitata. Viene quindi calcolata da RTE la velocità in base alla formula: iv = iv + ia * si Se la velocità risulta maggiore della velocità massima parametrizzata nelle variabili predefinite max_spe(n) e l'allarme spam è abilitato, viene emesso l'allarme velocità. Viene quindi calcolata da RTE la posizione in base alla formula = ip + iv * si Se la posizione risulta maggiore/minore dei finecorsa software parametrizzati nelle variabili predefinite max_str(n) e min_str(n) e la variabile predefinita cam è abilitata, viene emesso l' allarme crash (lower e upper). L'allarme servo (inseguimento) viene gestito se abilitato tramite la variabile predefinita sam
end_motion	End of the *motion* block
aux	In the *aux* fields of each RULE typically all those boundary conditions necessary to manage the process are expressed. The order in which the *aux*, *motion* blocks are written is free and determines their order of execution
end_aux	End of *aux* block
Validità	Rule
Note	The *ref* block, if any, must be the first block in the rule. There is no limit to the number of *aux* blocks within a rule block. You cannot have more than one *motion* block, nor more than one *ref* block.

© 2025 Robox SpA

Syntax	**RULE num** **axes** n[,n,n,...,n] | **axes_m** m **ref** ; REF BLOCK (optional) **end_ref** **motion** **if**(first_time()) ; only the first time **endif** ; MOTION LOCK **end_motion** **aux** ; AUX BLOCK (optional) **end_aux** **END_RULE**
	The declaration of formulas will use the normal instructions of the R3 language

Examples

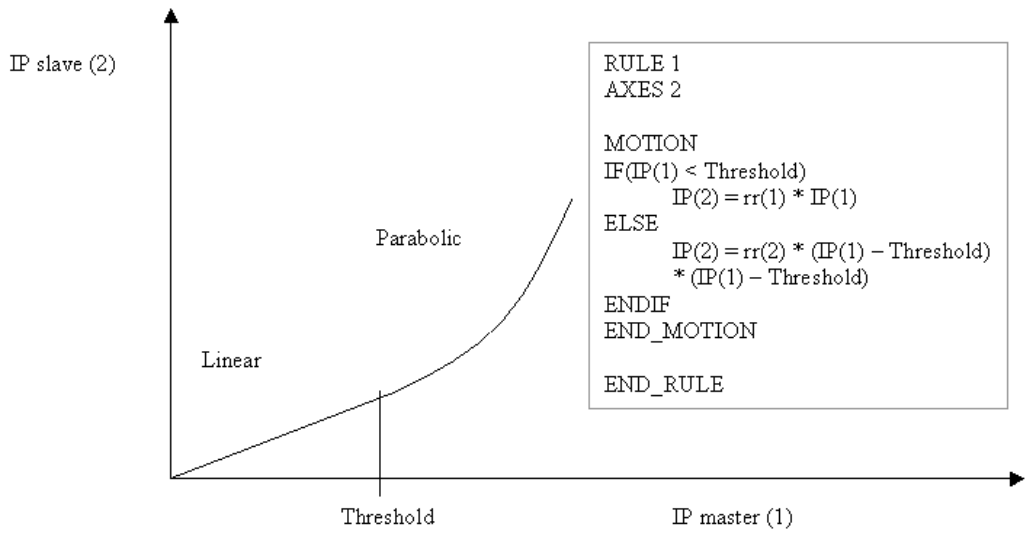
Example of use:

```
rule 7
axes 653 (1)
ref 717
    PASSA_1 = epos(1) * pro_gai(1)
    ; Analog output as a function of position error and theoretical
axis speed.
    sref(1) = PASSA_1 + iv(1) * kff(1)
end_ref 681
motion 703
    ; ...
end_motion 681
end_rule 682

rule 10
axes (2)
ref
    ; the reference voltage is equal to the position error multiplied
by kp.
    sref(2) = epos(2) * pro_gai(2)
end_ref

motion
    ; definition of trajectory type
 ; the ideal quota of axis 2 must be that of axis 1
 ; multiplied by the value contained in the RR(7) register.
    ip(2) = rr(7) * cp(1)
end_motion
```

© 2025 Robox SpA

```
end_rule
```

Example of use:




```
Errata corrige: ip(2) = (rr(1)*soglia) + (rr(2) * (ip(1)-soglia) *
(ip(1)-soglia))
; To construct the trajectory means to establish
; instant by instant the ideal position of a given axis.
; The trajectory is computed realtime (si) so it results in the
ability to
; implement any kind of law of motion.
```

Instruction rule_epilogue

With this instruction, it is possible to specify the name of a function that will be executed at each sampling of the rules 720 (which run at a fixed frequency set by the rule_freq instruction) after the execution of the ref 717 fields and after the executions of the motion/aux 703 fields.

Syntax	**rule_epilogue functionName**
functionName	It is the name of the function to be sent to execution (with no parameters and no return value)
Validity	Rule
Notes	Instruction must be called only once. If the instruction is called with *functionName* = 0 it is disabled

Examples

Example of use:

```
function 688 rule_init()
; We enable execution of rule_prologue in the initialization rule
rule_epilogue(myRuleName)
...
end_fun 680
; rule_epilogue function
function 688 myRuleName
...
end_fun 680
```

Instruction rule_freq

Function that allows the rule 720 file execution frequency($RULE 770) to be imposed from the program.

Syntax	**rule_freq (int freq)**
freq	Value containing the frequency (25 Hz -:- 5000 Hz) (optional, default 100Hz - 10 ms) (25 Hz = 40 ms; 1KHz = 1 ms; 2KHz = 500 micro seconds)
Validity	Rule, Task
Note	

Management of execution frequency rules.

There are three ways to impose the execution frequency (interrupt) of the RULES task:

- If CANbus and/or EtherCAT networks are being managed, the rules execution frequency is set equal to the PDO exchange frequency set in the respective fieldbus configuration editors. If both networks are being managed, the frequency must be the same in both editors.
- If no fieldbuses are being managed, the frequency is imposed from project configurator (see SYS_FREQ). If the "System frequency cannot be changed" flag is also present, a frequency change from program via the rule_freq instruction will not be possible and an attempt to use this instruction will generate an alarm.

Note: If the rules frequency set in the configurator is different from the PDO exchange frequency of the canbus and/or EtherCAT networks, the setting from the project configurator is ignored and a warning is given in the report.

- If no fieldbuses are handled and the "System frequency not changeable" flag is not entered in the project configurator, the rule_freq instruction can be used to change the frequency.

Examples

Examples of use:

```
; la frequenza delle rule viene impostata a 200 Hz
```

© 2025 Robox SpA

```
rule_freq(200)
; la frequenza delle rule viene impostata al valore contenuto nel
registro intero r(1)
rule_freq(r(1))
; definisco la costante VALORE_FREQ e le assegno il valore 400
lit 702 VALORE_FREQ 400
; la frequenza delle rule viene impostata al valore della costante
VALORE_FREQ
; quindi a 400 Hz
rule_freq(VALORE_FREQ)
```

Instruction rule_init

This special function, if defined by the user within the rules file($RULE 770), is automatically called by RTE once at startup, before the first rules execution.

Syntax	**function rule_init()** *; user code* ... **end_fun**
Validity	Rule
Note	The function is executed only once. Useful for performing initializations

Examples

Standard implementation of RULE_INIT():

```
function 688 rule_init()
    ; Definition of rules sampling frequency
 rule_freq 729 (500)
 ; Definition of prologue and epilogue functions
 rule_prologue 732 (myRulesPrologue)
 rule_epilogue 728 (myRulesEpilogue)
 ref_prologue 718 (myRefprologue)
 ;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 ; Insert here all initializations
 ; of the local variables used in the rule
    ;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
end_fun 680
```

Instruction rule_periodic

Instruction that allows a rule to be repeatedly sent to execution every "period" imposed. The rule must be written in an appropriate file beginning with the metacommand $RULE_PERIODIC 770. This rule has lower priority than synchronous rules 720 and, as a result, the period may fluctuate (gitter).

© 2025 Robox SpA

Syntax	**rule_periodic (i32 period, stru_int struct)**
period	Time interval between two successive executions [1 -:- 1,000,000 ms]
struct	Structure of type stru_int 842 containing related information (optional)
Validity	Rule, Task
Note	Instruction must be called only once. Instruction must be called again to change the period. If the instruction is called with period value 0 it is disabled

Diagnostics

- 2802 Interrupt rule - unknown number < STRU_INT.nRule>
- 9900 ill.Arg A:1 T:x ST:xxx If the period is not in the specified range [0 -:- 1,000,000 ms]
- 9900 ill.Arg A:2 T:x ST:xxx If the asynchronous rule task ($rule_int) has not been loaded into RAM (To load write command "load /fa/mytask.cof" in autoexec.stp)
- 9114 Rule Periodic : too long If code execution is too long.
- 9115 Rule Periodic : reduced freq.

Examples

You want to create a task that switches the state of a digital output every 5 minutes:

In the file mytask.r3:

```
$rule_periodic
rule 720 1
    out(257) = not 782 out(257)
end_rule 682
```

In the task:

```
; I define the variable parPeriodic of type STRU_INT 842
STRU_INT 842 parPeriodic
; continuous execution
parPeriodic.cntr.0 = 0
; totalizer is enabled to count
parPeriodic.cntr.1 = 0
; rule present in the file mytask.r3
parPeriodic.nrule = 1
; you do not want the cp picture
parPeriodic.maskPhoto = 0x0
...
; periodic rule enable condition
if(rise(inp(2)))
 ; enable execution rule periodic
 rule_periodic(300000 , parPeriodic)
```

© 2025 Robox SpA

```
endif
```

Instruction rule_prologue

With this instruction, it is possible to specify the name of a function that will be executed at each sampling of the rules 720 (which run at a fixed frequency set via the rule_freq 729 instruction) after the execution of the ref 717 fields but before the execution of the motion/aux 703 fields.

Syntax	**rule_prologue functionName**
functionName	It is the name of the function to be sent to execution (without parameters and without return value)
Validity	Rule
Note	Instruction must be called only once. If the instruction is called with functionName = 0 it is disabled

Examples

Example of use:

```
function 688 rule_init()
    ; We enable rule_prologue execution in the initialization rule
 rule_prologue(myRulePrologue)
 ...
end_fun 680
; rule_prologue function
function 688 myRulePrologue
...
end_fun 680
```

Instruction rule_start_norc

This instruction is used with RPE, and you also want to have the rule_prologue function, or rule_epilogue, active anyway

Normally rule_int is executed on the first occurrence of one of the following instructions/keywords:

- group 694
- order 708
- rc

Using rule_start_norc RTE, even in the absence of the above occurrences, executes rule_init 730 and disables the execution of all rules 720.

Syntax	**rule_start_norc**
Validity	Task

Syntax	**rule_start_norc**
Note	Fields ref 717 and end_ref 717 by default are not updated if no rule is present. Wanting to reactivate the rules would require acting on the order instruction (previously active rules will be put back into execution)

Instruction sb0

This instruction is used to write the first byte (starting with the least significant byte) of the argument with sign extension.

Syntax	**sb0 (i8|i16|i32|u8|u16|u32|real arg) = i8 value**
arg	Variable in which *value* will be imposed
value	Value to be imposed
Validity	Rule, Task
Note	From RTE 33.9.0 accessing a non-existing byte returns error

Instruction sb1

This instruction is used to write the second byte (starting with the least significant byte) of the argument with sign extension.

Syntax	**sb1 (i16|i32|u16|u32|real arg) = i8 value**
arg	Variable in which the *value* value will be imposed
value	Value to be imposed
Validity	Rule, Task
Note	As of RTE 33.9.0 access to a non-existing byte returns error

Instruction sb2

This instruction is used to write the third byte (starting with the least significant byte) of the argument with sign extension.

Syntax	**sb2 (i32|u32|real arg) = i8 value**
arg	Variable in which *value* will be imposed
value	Value to be imposed
Validity	Rule, Task

© 2025 Robox SpA

Syntax	**sb2 (i32|u32|real arg) = i8 value**
Notes	From RTE 33.9.0 accessing a non-existing byte returns error

Instruction sb3

This instruction is used to write the fourth byte (starting with the least significant byte) of the argument with sign extension.

Syntax	**sb3 (i32|u32|real arg) = i8 value**
arg	Variable into which the *value* will be imposed
value	Value to be imposed
Validity	Rule, Task
Note	From RTE 33.9.0 accessing a non-existing byte returns error

Instruction sb4

This instruction is used to write the fifth byte (starting with the least significant byte) of the argument with sign extension.

Syntax	**sb4 (real arg) = i8 value**
arg	Variable in which the *value* value will be imposed
value	Value to be imposed
Validity	Rule, Task
Note	As of RTE 33.9.0 access to a non-existing byte returns error

Instruction sb5

This instruction is used to write the sixth byte (starting with the least significant byte) of the argument with sign extension.

Syntax	**sb5 (real arg) = i8 value**
arg	Variable in which the *value* value will be imposed
value	Value to be imposed
Validity	Rule, Task
Note	As of RTE 33.9.0 access to a non-existing byte returns error

© 2025 Robox SpA

Instruction sb6

This instruction is used to write the seventh byte (starting with the least significant byte) of the argument with sign extension.

Syntax	**sb6 (real arg) = i8 value**
arg	Variable in which the *value* value will be imposed
value	Value to be imposed
Validity	Rule, Task
Note	As of RTE 33.9.0 access to a non-existing byte returns error

Instruction sb7

This instruction is used to write the eighth byte (starting with the least significant byte) of the argument with sign extension.

Syntax	**sb7 (real arg) = i8 value**
arg	Variable in which the *value* value will be imposed
value	Value to be imposed
Validity	Rule, Task
Note	As of RTE 33.9.0 access to a non-existing byte returns error

Instruction sdw0

This instruction is used to write the first dword (starting with the least significant dword) of the argument with sign extension.

Syntax	**sdw0 (i16|i32|u16|u32|real arg) = i32 value**
arg	Dword argument
value	Value to be imposed in arg
Validity	Rule, Task
Note	Since RTE 33.9.0 accessing a non-existing dword returns an error

© 2025 Robox SpA

Instruction sdw1

This instruction is used to write the second dword (starting with the least significant one) of the argument with sign extension.

Syntax	**sdw1 (i32|u32|real arg) = i32 value**
arg	Dword argument
value	Value to be imposed in arg
Validity	Rule, Task
Note	Since RTE 33.9.0 accessing a non-existing dword returns an error

Instruction select

This instruction allows a group of instructions to be selected for execution depending on the value of a variable.

Syntax	

	SELECT (i32 var)
	  CASE const
	  ; Instruction R3
	  BREAK
	  CASE const
	  ; Instruction R3
	  BREAK
	  DEFAULT
	  ; Instruction R3
	END_SELECT

var	Variable that depending on its value selects which instruction group to execute
const	Its value is the numerical identifier of the instruction group. It must be an integer numeric constant, otherwise an alarm is generated at compilation
Validity	Rule, Task
Note	"Instruction group" means those instructions that are between the case identifier and the next break, or between the default identifier and the end_select 682 identifier (if the default identifier is present). It is possible to have multiple case identifiers associated with the same break; this means that if an instruction group is not terminated by break, subsequent instructions will also execute until the first break (or end_select 682 identifier). If the variable value does not satisfy any of the specified cases (i.e., it does not match any of the cost values associated with the case

© 2025 Robox SpA

	identifiers), the instructions in the default group are executed. This block can be omitted, and in that case no instruction is executed

Examples

Example of use with state machine:

```
;-------------------------------------------------------
; Operating mode-dependent management (rModoOperativo)
;-------------------------------------------------------
select(rModoOperativo)
      ; Power failure management
      case M_MIS_POWER
            call HndMisPower
            break
      ; Mode change wait management
      case M_IDLE
            call HndIdle
            break
      ; Zero cycle management
      case M_CICLO_ZERO
            call HndZeroCycle
            break
      ; Manual movement management
      case M_MANUALE
            call HndManual
            break
      ; Management of initial placements
      case M_POSIZ
            call HndPosiz
            break
      ; Automatic movement management
      case M_AUTO
            call HndAuto
            break
      ; Here I get there only if no previous case
      default
            call HndMisPower
end_select
```

Instruction ser_con

Imposes the characteristic parameters of the specified serial communication channel.

Syntax	**ser_con (i32 n, i32 baud, i32 parity, i32 bit, i32 stop)**
n	Serial channel number (1, 2, 3, 4)

© 2025 Robox SpA

baud	Baud rate value, to be chosen from the following: 110 - 150 - 300 - 600 - 1200 - 2400 - 4800 - 9600 - 19200 - 38400 - 57600 - 115200. **Note**: Since RTE 34.22.01, management has also been added for the following baud rates: 128000 - 256000 - 512000 - 1000000 - 2000000.
parity	Parity type, to be chosen from the following: • E (even parity) • O (odd parity) • N (no parity bit)
bit	Number of word bits (allowable value: 7, 8)
stop	Number of stop bits, to be chosen from the values 1 and 2
Validity	Rule, Task
Note	From RTE 34.36.1, serial communication parameters can also be imposed on ports communicating with BCC3 protocol

Diagnostics

- 9900 ill. Arg A:1 T:x ST:xxx if illegal serial number

Examples

Example of use:

```
; Serial port number 1 of the connected control is programmed with:
; frequency of 9600 Baud
; without parity
; 8 word bits
; 1 stop bit.
; Execution of this instruction changes the parameters of the
selected port.
; exclusively on the connected control.
; It does not, evidently, change those of the serial port of the PC
; possibly connected to the control itself.
ser_con(1, 9600, 'N', 8, 1)
```

Instruction ser_w_sts

Instruction to set the two signals DTR and RTS of the specified serial channel.

Syntax	**ser_w_sts (i32 nSer, i32 mask, i32 sts)**
nSer	Serial channel number

© 2025 Robox SpA

Syntax	**ser_w_sts (i32 nSer, i32 mask, i32 sts)**
mask	Mask to enable the bits of the status word to be changed. It can take the following values: <table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>No modification enabled</td></tr><tr><td>1</td><td>Enable modification for DTR signal</td></tr><tr><td>2</td><td>Enable modification for RTS signal</td></tr><tr><td>4</td><td>Enable modification for DTR and RTS signals</td></tr></table>
sts	It is the value of the DTR, RTS signals that are intended to be forced. Forcing a signal to 1 means bringing it to its active condition; therefore, we do not refer to the physical levels of the signals, but to the logical levels
Validity	Rule, Task
Note	You can only change the status of these signals if you have previously selected MODEM mode for the specified serial channel (see the ser_mod 566 instruction)

Examples

Example of use:

```
ser_w_sts 738(1, 1, 0)
; For serial 1, the DTR signal is forced to 0
; however, the RTS signal is not affected
```

Instruction set_float

This function writes a FLOAT to a buffer at the specified offset.

Syntax	**set_float (buffer, i32 offset, float set_val)**
buffer	Can be an array, structure, register
offset	Offset in bytes
set_val	FLOAT value to be written

© 2025 Robox SpA

Syntax	**set_float (buffer, i32 offset, float set_val)**
Validity	Task, Rule
Note	NO range check is performed, it is up to the user to check that adding the offset and the data size does not exceed the maximum buffer size. See example. If the user wishes to write a float into the 'data' buffer of a structure of type bbc_msguser 793 then he must use the function bcc_set_float(). The mirror function is the get_float() 325

Examples

Once you have loaded the example program on a control, via a shell you can verify that set_float works. To do this, set a value between 0 and 127 - sizeof(my_data) to the variable offset, then set a non-zero value to the variable set_val and verify that the variable my_data is equal to set_val.

```
u8 buffer[128]
i32 offset
float set_val
float my_data

; -------- loop begin --------
__MAIN_LOOP__

   ; range check
   if(offset >= 0 and (offset + sizeof(my_data)) < 128)
      set_float(buffer, offset, set_val)
      my_data = get_float(buffer, offset)
   endif

END_MAIN_LOOP
```

Instruction set_i16

This function writes a word I16 into a buffer at the specified offset.

Syntax	**set_i16 (buffer, i32 offset, i16 set_val)**
buffer	Can be an array, structure, or register
offset	Offset in bytes
set_val	The word to be written
Validity	Task, Rule
Note	NO range check is performed, it is up to the user to check that adding the offset and the

© 2025 Robox SpA

Syntax	**set_i16 (buffer, i32 offset, i16 set_val)**
	data size does not exceed the maximum buffer size. See example. If the user wishes to write a word into the 'data' buffer of a structure of type bbc_msguser 793 then he must use the function bcc_set_i16(). The mirror function is the get_i16() 328

Examples

Once the example program is loaded on a control, via a shell it is possible to verify the operation of set_i16. To do this, set a value between 0 and 127 - sizeof(my_data) to the variable offset, then set a non-zero value to the variable set_val and verify that the variable my_data is equal to set_val.

```
u8 buffer[128]
i32 offset
i16 set_val
i16 my_data

; -------- loop begin --------
__MAIN_LOOP__

   ; range check
   if(offset >= 0 and (offset + sizeof(my_data)) < 128)
      set_i16(buffer, offset, set_val)
      my_data = get_i16(buffer, offset)
   endif

END_MAIN_LOOP
```

Instruction set_i32

This function writes an i32 variable into a buffer at the specified offset.

Syntax	**set_i32 (buffer, i32 offset, i32 set_val)**
buffer	Can be an array, a structure, or a register
offset	Offset in bytes
set_val	The I32 value to be written
Validity	Task, Rule
Note	NO range check is performed, it is up to the user to check that adding the offset and the data size does not exceed the maximum buffer size. See example. If the user wishes to write an i32 into the 'data' buffer of a structure of type

© 2025 Robox SpA

Syntax	**set_i32 (buffer, i32 offset, i32 set_val)**
	bbc_msguser 793 then he must use the function bcc_set_i32(). The mirror function is the get_i32() 328

Examples

Once you have loaded the example program on a control, via a shell you can verify the operation of set_i32. To do this, set a value between 0 and 127 - sizeof(my_data) to the offset variable, then set a non-zero value to the R(1) register and verify that the variable my_data is equal to R(1).

```
u8 buffer[128]
i32 offset
i32 my_data

; -------- loop begin --------
__MAIN_LOOP__

   ; range check
   if(offset >= 0 and (offset + sizeof(my_data)) < 128)
      set_i32(buffer, offset, r(1))
      my_data = get_i32(buffer, offset)
   endif

END_MAIN_LOOP
```

Instruction set_i64

This function writes into a buffer, at the specified offset, the value of a stru_large_int structure.

Syntax	**set_i64 (buffer, i32 offset, stru_large_int data)**
buffer	Can be an array, structure, register
offset	Offset in bytes
data	Variable of type stru_large_int to be written
Validity	Task, Rule
Note	The mirror function is get_i64() 692

Examples

Once you have loaded the example program on a control, via a shell you can verify that set_i64 works. To do this, set a value between 0 and 127 - sizeof(my_data) to the offset variable, then set a non-zero value to the variables val_high and val_low, then verify that the structure my_data is equal to set_val.

© 2025 Robox SpA

```
u8 buffer[128]
i32 val_high
u32 val_low
stru_large_int set_val
stru_large_int my_data

; -------- loop begin --------
__MAIN_LOOP__

   ; range check
   if(offset >= 0 and (offset + sizeof(my_data)) < 128)
      set_val.high = val_high
      set_val.low  = val_low
      set_i64(buffer, offset, set_val)
      get_i64(buffer, offset, my_data)
   endif

END_MAIN_LOOP
```

Instruction set_i8

This function writes a variable i8 into a buffer at the specified offset.

Syntax	**set_i8 (buffer, i32 offset, i8 set_val)**
buffer	Can be an array, structure, or register
offset	Offset in bytes
set_val	The byte to be written
Validity	Task, Rule
Note	Range check is NOT performed, it is up to the user to check that adding the offset and the data size does not exceed the maximum buffer size. See example. If the user wishes to write a byte into the 'data' buffer of a structure of type bbc_msguser 793 then he must use the function bcc_set_i8(). The mirror function is the get_i8() 329

Examples

Once you have loaded the example program on a control, via a shell you can verify the operation of set_i8. To do this, set a value between 0 and 127 - sizeof(my_data) to the variable offset, then set a non-zero value to the variable set_val and verify that the variable my_data is equal to set_val.

```
u8 buffer[128]
i8 set_val
```

© 2025 Robox SpA

```
i32 offset
i8 my_data

; -------- loop begin --------
__MAIN_LOOP__

   ; range check
   if(offset >= 0 and (offset + sizeof(my_data)) < 128)
      set_i8(buffer, offset, set_val)
      my_data = get_i8(buffer, offset)
   endif

END_MAIN_LOOP
```

Instruction set_real

This function writes a REAL to a buffer at the specified offset.

Syntax	**set_real (buffer, i32 offset, real set_val)**
buffer	Can be an array, structure, register
offset	Offset in bytes
set_val	The real value to be set in the buffer
Validity	Task, Rule
Note	range check is NOT performed, it is up to the user to check that adding the offset and the data size does not exceed the maximum buffer size. See example. If the user wishes to write a real into the 'data' buffer of a structure of type bbc_msguser 793 then he must use the function bcc_set_double(). The mirror function is the get_real() 330

Examples

Once you have loaded the example program on a control, via a shell you can verify that set_real works. To do this, set a value between 0 and 127 - sizeof(my_data) to the variable offset, then set a non-zero value to the variable set_val and verify that the variable my_data is equal to set_val.

```
u8 buffer[128]
i32 offset
real set_val
real my_data

; -------- loop begin --------
__MAIN_LOOP__
```

```
   ; range check
   if(offset >= 0 and (offset + sizeof(my_data)) < 128)
      set_real(buffer, offset, set_val)
      my_data = get_real(buffer, offset)
   endif

END_MAIN_LOOP
```

Instruction set_u16

This function writes a U16 word to a buffer at the specified offset.

Syntax	**set_u16 (buffer, i32 offset, u16 set_val)**
buffer	Can be an array, structure, or register
offset	Offset in bytes
set_val	The word to be written
Validity	Task, Rule
Note	Range check is NOT performed, it is up to the user to check that adding the offset and the data size does not exceed the maximum buffer size. See example. If the user wishes to write a word into the 'data' buffer of a structure of type bbc_msguser 793 then he must use the function bcc_set_u16(). The mirror function is the get_u16() 331

Examples

Once you have loaded the example program on a control, via a shell you can verify that set_u16 works. To do this, set a value between 0 and 127 - sizeof(my_data) to the offset variable, then set a non-zero value to inp_w 200 and verify that the variable my_data is equal to inp_w 200.

```
u8 buffer[128]
i32 offset
u16 my_data

; -------- loop begin --------
__MAIN_LOOP__

   ; range check
   if(offset >= 0 and (offset + sizeof(my_data)) < 128)
      set_u16(buffer, offset, inp_w(200))
      my_data = get_u16(buffer, offset)
   endif
```

© 2025 Robox SpA

```
END_MAIN_LOOP
```

Instruction set_u32

This function writes a u32 variable into a buffer at the specified offset.

Syntax	**set_u32 (buffer, i32 offset, u32 set_val)**
buffer	Can be an array, structure, or register
offset	Offset in bytes
set_val	U32 value to be written
Validity	Task, Rule
Note	Range check is NOT performed, it is up to the user to check that adding the offset and the data size does not exceed the maximum buffer size. See example. If the user wishes to write a u32 into the 'data' buffer of a structure of type bbc_msguser 793 then he must use the function bcc_set_u32(). The mirror function is the get_u32() 332

Examples

Once you have loaded the example program on a control, via a shell you can verify the operation of set_u32. To do this, set a value between 0 and 127 - sizeof(my_data) to the variable offset, then set a non-zero value to the variable set_val and verify that the variable my_data is equal to set_val.

```
u8 buffer[128]
u32 offset
u32 set_val
u32 my_data

; -------- loop begin --------
__MAIN_LOOP__

   ; range check
   if(offset >= 0 and (offset + sizeof(my_data)) < 128)
      set_u32(buffer, offset, set_val)
      my_data = get_u32(buffer, offset)
   endif

END_MAIN_LOOP
```

Instruction set_u64

This function writes into a buffer, at the specified offset, the value of a stru_large_int structure.

Syntax	**set_u64 (buffer, i32 offset, stru_large_uint data)**
buffer	Can be an array, structure, register
offset	Offset in bytes
data	Variable of type stru_large_int in which to save the read value
Validity	Task, Rule
Note	The mirror function is the get_u64() 693

Examples

Once you have loaded the example program on a control, via a shell you can verify that the get_u64 works. To do this, set a value between 0 and 127 - sizeof(my_data) to the offset variable, then set a non-zero value to the variables val_high and val_low, then verify that the my_data structure is equal to set_val.

```
u8 buffer[128]
u32 val_high
u32 val_low
stru_large_uint set_val
stru_large_uint my_data

; -------- loop begin --------
__MAIN_LOOP__

   ; range check
   if(offset >= 0 and (offset + sizeof(my_data)) < 128)
      set_val.high = val_high
      set_val.low  = val_low
      set_u64(buffer, offset, set_val)
      get_u64(buffer, offset, my_data)
   endif

END_MAIN_LOOP
```

Instruction set_u8

Questa funzione scrive una variabile u8 in un buffer all'offset specificato.

Syntax	**set_u8 (buffer, i32 offset, u8 setVal)**
buffer	Can be an array, structure, or register
offset	Offset in bytes

© 2025 Robox SpA

Syntax	**set_u8 (buffer, i32 offset, u8 setVal)**
setVal	The byte to be written
Validity	Task, Rule
Note	No range check is performed; it is up to the user to check that adding the offset and the data size does not exceed the maximum buffer size. See example. If the user wishes to write a byte into the 'data' buffer of a structure of type bbc_msguser 793 then he must use the instruction bcc_set_u8 661(). The mirror function is the get_u8 333()

Examples

Once you have loaded the example program on a control, via a shell you can verify the operation of set_u8. To do this, set a value between 0 and 127 - sizeof(my_data) to the variable offset, then set a non-zero value to the variable set_val and verify that the variable my_data is equal to set_val.

```
u8 buffer[128]
u8 set_val
i32 offset
u8 my_data

; -------- loop begin --------
__MAIN_LOOP__

   ; range check
   if(offset >= 0 and (offset + sizeof(my_data)) < 128)
      set_u8(buffer, offset, set_val)
      my_data = get_u8(buffer, offset)
   endif

END_MAIN_LOOP
```

Instruction set_version

Allows the software version of the application to be written to the memory area that is displayed by the "SYSINFO" command (to be sent from shell) and 4*20 viewers.

Syntax	**set_version (i32 version, i32 release, i32 build, string desc)**
version	Value containing the version number
release	Value containing the release number
build	Value containing the build number
descr	Descriptive text of 32 characters maximum

© 2025 Robox SpA

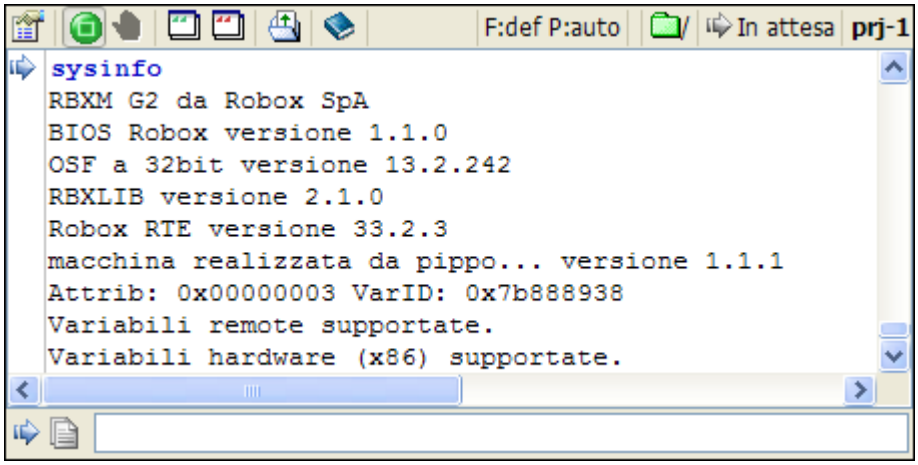
Syntax	**set_version (i32 version, i32 release, i32 build, string desc)**
Validity	Rule, Task
Note	

Examples

If I enter the instruction:

```
set_version(1,1,1,"machine made by....")
```

With sysinfo command:



Instruction sw0

This instruction is used to write the first word (starting with the least significant) of the argument with sign extension.

Syntax	**sw0 (i16|i32|u16|u32|real arg) = i16 value**
arg	Word argument
value	Value to be imposed
Validity	Rule, Task
Note	

Instruction sw1

This instruction is used to write the second word (starting with the least significant) of the argument with sign extension.

Syntax	**sw1 (i32|u32|real arg) = i16 value**
arg	Word argument

© 2025 Robox SpA

Syntax	**sw1 (i32|u32|real arg) = i16 value**
value	Value to be imposed
Validity	Rule, Task
Note	

Instruction sw2

This instruction is used to write the third word (starting with the least significant) of the argument with sign extension.

Syntax	**sw2 (real arg) = i16 value**
arg	Word argument
value	Value to be imposed
Validity	Rule, Task
Note	

Instruction sw3

This instruction is used to write the fourth word (starting with the least significant) of the argument with sign extension.

Syntax	**sw3 (real arg) = i16 value**
arg	Word argument
value	Value to be imposed
Validity	Rule, Task
Note	

Instruction terminator

This instruction imposes another active terminator in addition to the standard default LF terminator.

Syntax	**terminator (i32 num, i32 code)**
num	Locates the input device
code	Represents the AscII code of the terminator to be set
Validity	Rule, Task
Note	

Diagnostics

© 2025 Robox SpA

- Generate alarm 9900 ill. Arg A:1 T:x ST:xxx if illegal serial number

Examples

Example of use:

```
; Impose ETX (3) as terminator.
; Strings may be terminated by either ETX or LF.
terminator(1, 3)
; Imposes LF as the only terminator.
; Terminator is used only if in "normal mode" (see ser_mod).
terminator(1, 10)
```

Instruction tm_et

This instruction is used to set the current value (Elapsed Time) of the timer.

WARNING: The value can only be set before the timer is started (e.g., before calling the tm_int 751() function for the first time), otherwise alarm 9901 is generated.

Syntax	**tm_et (timer name, real valueEt)**
name	Name of the timer of type timer 900
valueEt	Value of the time, in milliseconds, to be imposed
Validity	Rule, Task
Note	

Instruction tm_int

This instruction acts as a timer to delay a rising edge by a predefined time.

Functioning

An integral timer is created.

The elapsed time (ET) starts evolution as soon as there is a rising edge of the *start* input and continues to increase as long as the input remains high.

The output (Q) goes into the ON state after the elapsed time (ET) reaches the specified preset time period (PT).

In contrast to tm_on 754(), the count is not reset by the absence of the input, but only after a reset command; thus the elapsed time is equal to the sum of the enable times.

Syntax	**tm_int (timer name, bool start)**
name	Name of the timer of type timer 900
start	Timer start input (rising edge).

© 2025 Robox SpA

Syntax	**tm_int (timer name, bool start)**
	Can be any condition, even an expression (e.g., cp(1) > 1000)
Validity	Rule, Task
Note	Timers are not retentive

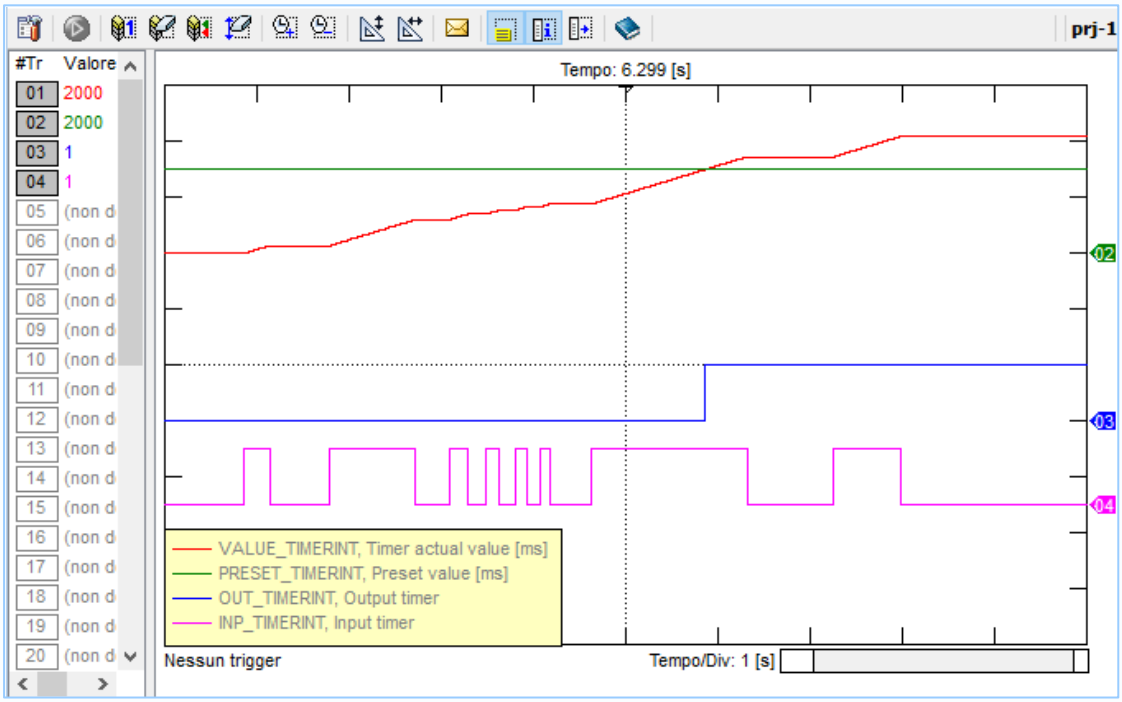
Diagnostics

- Generate alarm 9901 ill. fun. if the timer is previously used with another type of timer

Examples

Example of use:

```
timer 900 T1
tm_pt 617(T1, 1000) ; we set the preset time to 1 sec
tm_int 751(T1, inp(1)) ; if inp 1 true the timer increments the elapsed
time
out(257) = tm_q 619(T1) ; arrived at the preset value the result goes
ON
et = tm_et 751(T1) ;  elapsed time out
_if(inp(2))
 tm_reset 756(T1) ; resets the elapsed time
```



© 2025 Robox SpA

Instruction tm_of

This instruction acts as a timer to delay a falling edge by a predefined time.

Functioning

On the rising edge of the *start* input, the output (Q) goes into the ON state.

The elapsed time (ET) begins evolution as soon as there is a falling edge of the *start* input and continues to increase until the rising edge of the start input, at which time it is imposed equal to 0.

The output (Q) remains in the ON state until the elapsed time (ET) reaches the specified preset time period (PT).

Syntax	**tm_of (timer name, bool start)**
name	Name of the timer of type timer 900
start	Timer start input (rising edge). Can be any condition, even an expression (e.g., cp(1) > 1000)
Validity	Rule, Task
Note	Timers are not retentive

Diagnostics

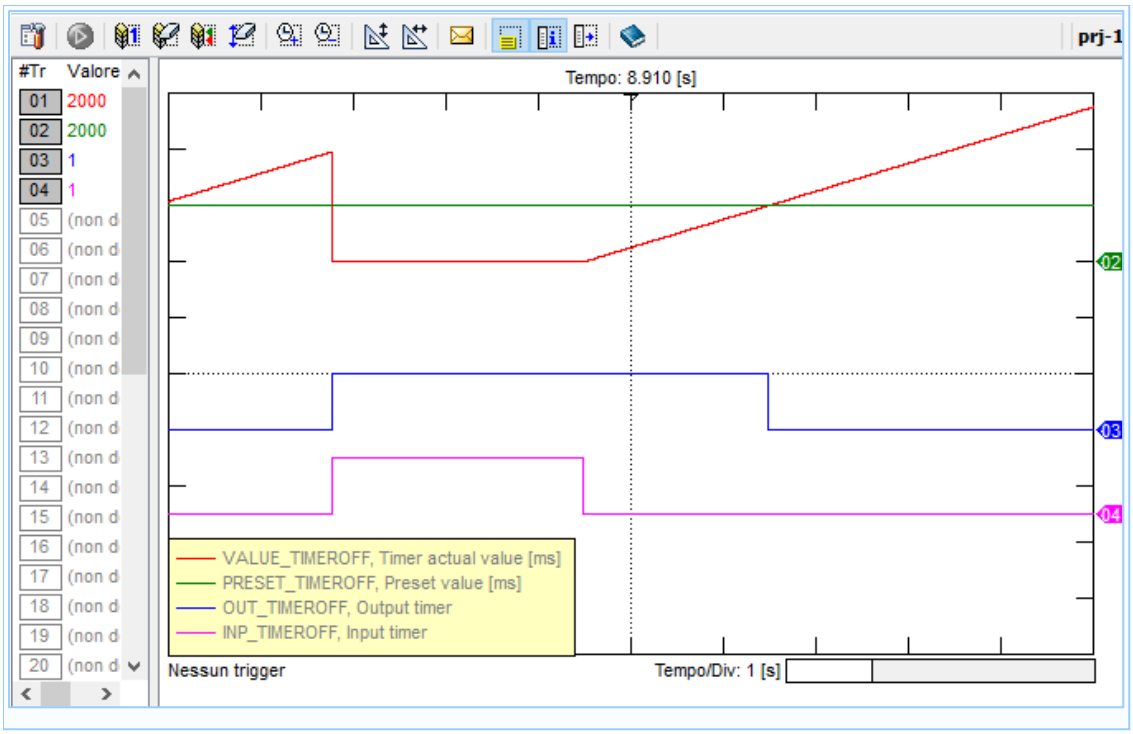
- Generate alarm 9901 ill. fun. if the timer is previously used with another type of timer

Examples

Example of use:

```
timer 900 T1
tm_pt 617(T1, 1000) ; we set the preset time to 1 sec
tm_of 753(T1 , inp(1)) ; on rise input 1 (ADV button on CPU) outcome of
timer goes ON
out(257) = tm_q 619(T1) ; result of timer goes ON for 1 sec in the
absence of 'input 1
et = tm_et 751(T1) ; elapsed time
```

© 2025 Robox SpA



Instruction tm_on

This instruction acts as a timer to delay a rising edge by a predefined time.

Functioning

The elapsed time (ET) starts evolution as soon as there is a rising edge of the *start* input and continues to increase until the falling edge of the start input, at which time it is imposed equal to 0.

The output (Q) goes into the ON state after the elapsed time (ET) reaches the specified preset time period (PT).

Syntax	**tm_on (timer name, bool start)**
name	Name of the timer of type timer 900
start	Timer start input (rising edge). Can be any condition, even an expression (e.g., cp(1) > 1000)
Validity	Rule, Task
Note	Timers are not retentive. In case of programmatic approach tm_on() should be initialized and set to false before the main loop

Diagnostics

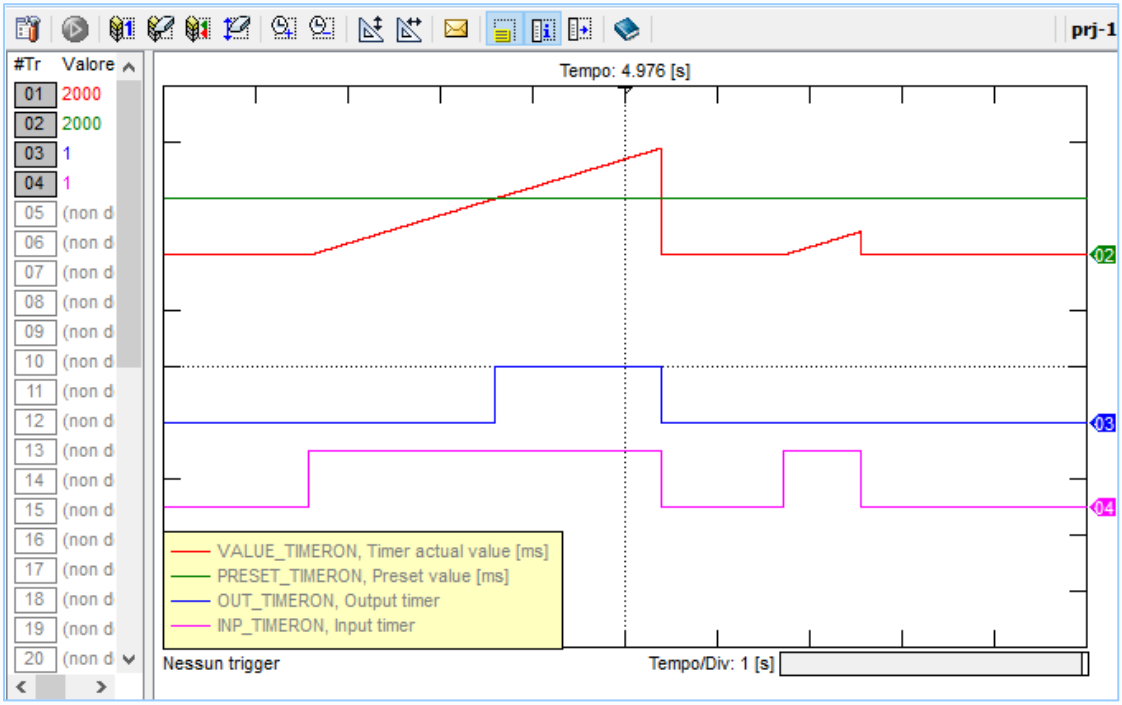
© 2025 Robox SpA

- Generate alarm 9901 ill. fun. if the timer is previously used with another type of timer

Examples

Example of use:

```
timer 900 T1
tm_pt 617(T1, 1000) ; we set the preset time to 1 sec
tm_on 754(T1, inp(1)) ; on rise input 1 (ADV button on CPU) we start
counting
out(257) = tm_q 619(T1) ; we activate output 257 when the result of the
timer goes ON 1 sec later
et = tm_et 751(T1) ; elapsed time
```



Example usage:

NOTE: In case of programmatic approach tm_on() should be initialized and set to false before the main loop.

```
timer T1
tm_pt(T1, 2000) ; we set  the preset time to 2 sec.
tm_on(T1, false) ; we set to false
; =============
; MAIN LOOP
; =============
```

Instruction tm_pt

This instruction is used to impose the preset value of the timer.

Syntax	**tm_pt (timer name, real presetTm)**
name	Name of the timer of type timer 900
presetTm	Value of the preset, in milliseconds, to be imposed
Validity	Rule, Task
Note	

Instruction tm_pt2

This instruction is used to impose the value of the second preset of the square timer.

Syntax	**tm_pt2 (timer name, real presetTm)**
name	Name of the timer of type timer 900
presetTm	Value of the preset, in milliseconds, to be imposed
Validity	Rule, Task
Note	

Instruction tm_reset

This instruction is used to reset the time totalizer (ET) of a timer.

It can be usefull if you are using timers of integral type tm_int 751().

Syntax	**tm_reset (timer name)**
name	Name of the timer of type timer 900
Validity	Rule, Task
Note	

Instruction tm_sqw

This instruction acts as a timer to generate a square wave.

Functioning

It generates a square wave output of which the user imposes the duration of the high (Q=1) and low (Q=0) states via (PT) and (PT2), as long as the enable signal (EN) is set to 1; when this signal is set to 0, the timer will stop until the next state change.

© 2025 Robox SpA

Syntax	**tm_sqw (timer name, bool start)**
name	Name of the timer of type timer 900
start	Start input of the timer (rising edge). Can be any condition, even an expression (e.g., cp(1) > 1000)
Validity	Rule, Task
Note	Timers are not retentive

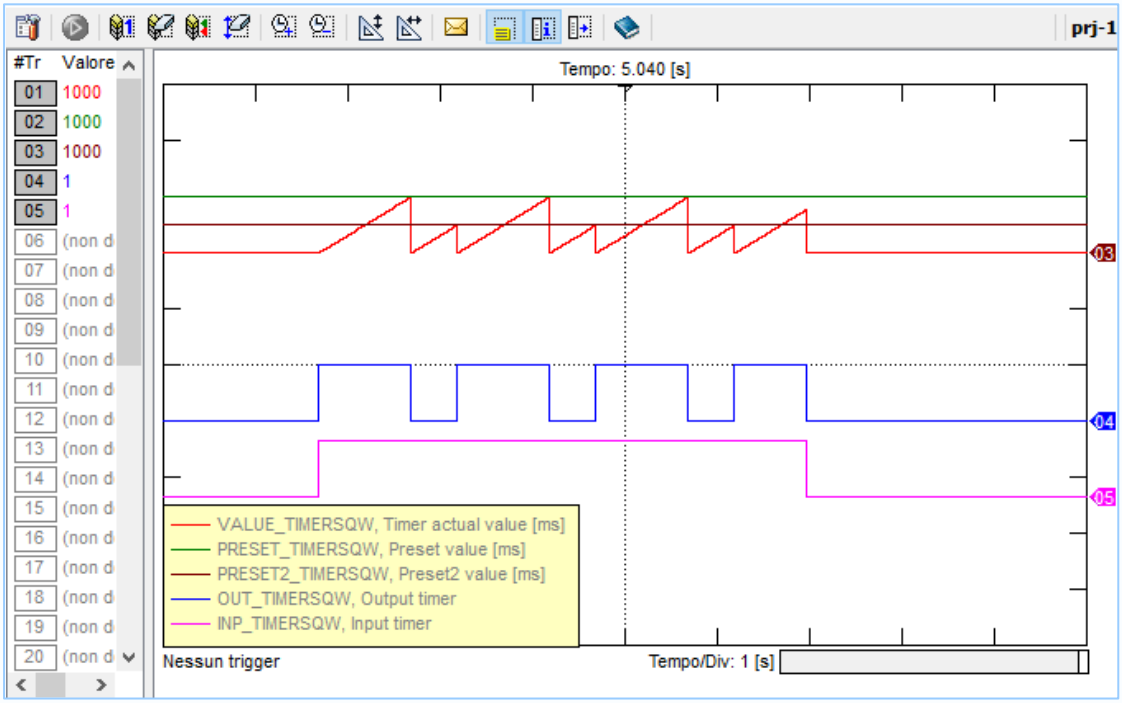
Diagnostics

- Generate alarm 9901 ill. fun. if the timer is previously used with another type of timer

Examples

Example of use:

```
timer 900 T1
tm_pt 617(T1, 1000) ; we set the preset time to 1 sec
tm_pt2 756(T1, 2000) ; we set  the second preset time to 2 sec
tm_sqw 756(T1, inp(1)) ; on rise input 1 (ADV button on CPU) we start
counting
out(257) = tm_q 619(T1) ; we activate for 1 sec the output 257
et = tm_et 751(T1) ;  elapsed time
```



© 2025 Robox SpA

Instruction tm_tp

This instruction acts as a timer to generate a pulse of predefined duration.

Functioning

On the rising edge of the *start* input, output Q remains in the ON state for the specified preset time period (PT) even if the input should experience a new rising edge.

The elapsed time (ET) begins evolution as soon as there is a rising edge of the *start* input and continues to increase until the falling edge of the input itself, at which time it is imposed equal to 0 (however, not before the time (PT) is over).

Syntax	**tm_tp (timer name, bool start)**
name	Name of the timer of type timer 900
start	Start input of the timer (rising edge). Can be any condition, even an expression (e.g., cp(1) > 1000)
Validity	Rule, Task
Note	Timers are not retentive

Diagnostics

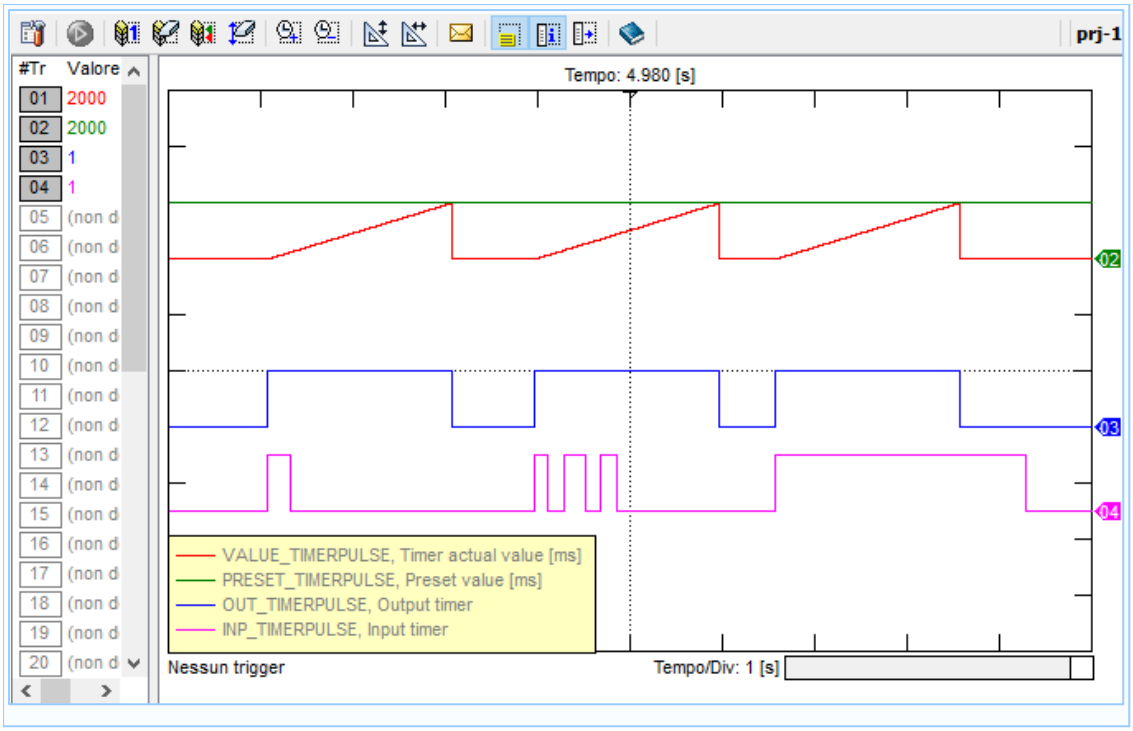
- Generate alarm 9901 ill. fun. if the timer is previously used with another type of timer

Examples

Example of use:

```
timer 900 T1
tm_pt 617(T1, 1000) ; we set the preset time to 1 sec
tm_tp 758(T1, inp(1)) ; on the high state of 'input 1 (ADV button on
CPU) the elapsed time is incremented
out(257) = tm_q 619(T1) ; the result of the timer stays ON for 1 second
et = tm_et 751(T1) ;  elapsed time
```

© 2025 Robox SpA



Instruction w0

This instruction is used to write the first word (starting with the least significant) of the argument.

Syntax	**w0 (i16|i32|u16|u32|real arg) = u16 value**
arg	Word argument
value	Value to be imposed
Validity	Rule, Task
Note	This instruction cannot be used to set the value of a register (e.g., w0(r(1))=val)

Instruction w1

This instruction is used to write the second word (starting with the least significant) of the argument.

Syntax	**w1 (i32|u32|real arg) = u16 value**
arg	Word argument
value	Value to be imposed
Validity	Rule, Task

© 2025 Robox SpA

Syntax	**w1 (i32|u32|real arg) = u16 value**
Note	This instruction cannot be used to set the value of a register (e.g., w1(r(1))=val)

Instruction w2

This instruction is used to write the third word (starting with the least significant) of the argument.

Syntax	**w2 (real arg) = u16 value**
arg	Word argument
value	Value to be imposed
Validity	Rule, Task
Note	This instruction cannot be used to set the value of a register (e.g., w2(r(1))=val)

Instruction w3

This instruction is used to write the fourth word (starting with the least significant) of the argument.

Syntax	**w3 (real arg) = u16 value**
arg	Word argument
value	Value to be imposed
Validity	Rule, Task
Note	This instruction cannot be used to set the value of a register (e.g., w3(r(1))=val)

Instruction while

This construct allows the execution of the group of instructions between while and end_while to be repeated as long as the condition defined in the instruction holds true.

Syntax	**while (condition)** *; Instruction R3* . . . **end_while**
condition	Can be a generic mathematical expression, the result of which is taken as false if it is worth 0, otherwise as true
Validity	Rule, Task

© 2025 Robox SpA

Syntax	**while (condition)** *; Instruction R3* ... **end_while**
Note	The use of the break is allowed

Examples

Example of use with condition:

```
; start of while construct ... end_while
; the loop is repeated until the value of r(3) is less than 10
while(r(3) < 10)
 ; instruction R
 ...
 r(3) = r(3) + 1
; end of while construct ... end_while
end_while 683
```

Example of infinite loop:

```
; infinite loop
while(1)
 ; instruction R
end_while 683.
```

Example of use with break:

```
while(1)
 ; instruction R
 ...
 if 696 (HAND)
 break 667
 endif 683
 ; if manual is true [<> 0] here we never pass!
 ...
end_while 683
```

Grammar

Grammar generalities

An R3 program is written in a free format. Lines of instructions can be written in columns as preferred, with any number of spaces.

Each line, separated from another line by the characters CR LF (Line Feed-ENTER key), must contain only one instruction. An instruction may be extended over several lines as long as the character "&" (ampersand) is inserted at the end of each line to indicate to the compiler that the instruction is not finished. An "&" character is treated as an instruction continuation only if it is at the end of the line (not followed by any character). Otherwise, "&" is treated as character being part of the instruction.

© 2025 Robox SpA

The ";" (semicolon) character introduces a comment: a line preceded by this character remains in the listing, but is ignored by the R3 compiler.

Example:

```
; single-line instruction
r(27) = 123
```

Example:

```
; same instruction extended over two lines
r(27) = &
123
```

A program written in the R3 language appears as a sequence of sentences that must be written respecting certain grammatical and syntactic rules.

In these sentences will appear words reserved for the language itself (keywords), variable names, names defined by the user, and obviously numbers.

The set of characters that are accepted by the R3 compiler, for keywords and variable names, is as follows:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

a b c d e f g h i j k l m n o p q r s t u v w x y z

0 1 2 3 4 5 6 7 8 9

Plus some special characters:

= . + - * / , : [ ] ( ) " $ _ ; & # tabulator space

The following special characters are reserved for Robox developments:

@ % ! ^ ' ? | \ ~

NOTE: The compiler makes no difference between lowercase and uppercase letters for keywords and variable names (case-insensitive).

Example:

```
; These three words are correct and equivalent
OUTPUT
Output
oUtPuT
```

Identifiers

Identifiers are used to name variables, subroutines, jump destinations (labels).

The first character of an identifier must be alphabetic. Subsequent characters may be alphabetic or numeric.

In addition, the use of the underscore (_) is permitted. There are no special limits on the number of characters that can be used.

Example:

```
; declaration of the identifier PIPPO associated with an integer
variable.
INT pippo
; declaration of the PLUTO identifier associated with a label
(subroutine name or jump destination)
pluto:
```

© 2025 Robox SpA

NOTE: R3 keywords cannot be used as identifiers.

Introduction

R3 language generalities

Flexible automation has found in programming languages its most powerful means of expression. These languages have become especially popular for programming machine tools, flexible transfers, robots, automatic machines, etc.

However, there is a whole category of machines, characterized by many axes that rotate continuously and must be synchronized according to more or less complicated laws of motion, which has not yet been able to conveniently enjoy the flexibility characteristic of today's automation. We refer, for example, to the following fields of application:

- printing machines
- textile machines
- metalworking machines
- paper processing machines
- woodworking machines
- in general synchronism controls

Mechanical solutions still dominate in these fields, such as, for example:

- cams of appropriate profile for storing a certain law of motion
- drums with cams and micros for storing programs
- mechanically made synchronisms, i.e., with main shafts from whose movement, by means of gears, all movements for the machine are derived
- more or less ingenious linkages
- etc.

Electronic solutions are then, normally, analog, or derived from analog, unless the digital solution is indispensable; for example, in the synchronism control of two slave motors.

Evidently, under these conditions, the flexibility of the systems, that is, the ability to adapt to new production requirements, is very low and, in any case, linked to time-consuming and, above all, expensive mechanical operations. Costly in itself and costly above all in terms of lost production.

Similarly, centralized management of production in both its managerial and diagnostic aspects turns out to be impossible.

Goals

Robox SpA, building on its experience in motion control gained in robotics, has realized the R3 language for programming the work cycles of these machines.

The R3 language offers many advantages, such as:

- returns usable Robox motion controller, developed for robotics, to the more general field of motion control and synchronisms on axes that can also rotate continuously
- it provides software solution to motion problems
- it allows easy integration of the machines thus controlled into the "automatic factory"

© 2025 Robox SpA

These machines, in general, are characterized by the number of controlled axes, axis being understood to mean a part of the machine whose motion can be controlled independently by an actuator.

The actuator will typically be a CC motor, a brushless motor, an hydraulic rotary motor, but also an induction motor, an hydraulic cylinder, etc.

A typical example is that of a coordinate transformation program for one or more axes, which must perform a series of calculations at a fixed rate, e.g., every 2 ms, taking the information from the master processor and returning the results in real time.

Advantages of languages

Solving problems by programming, that is, by writing software, offers countless advantages.

We recall the main ones:

- the description of problems, in this case laws of motion and operating cycles, is fast, simple and documented
- the time between the beginning of the design of a machine and fine-tuning is drastically reduced
- mechanical parts of machines have a much higher degree of standardization
- mechanical parts that are difficult to manifacture are eliminated
- machine reliability increases, in conjunction with a decrease in mechanical complications
- design errors are quickly and easily eliminated
- it is very easy to optimize machine modes of operation and thus improve operation or adapt to new requirements
- it is very easy to introduce modifications

Practically there are no disadvantages. The only potential disadvantage that may still exist is the necessity of having to change the way of designing and thus having to invest time in learning the new tools of the trade: computers, programming languages, editors, serial transmissions, compilers, etc.; as well as having to assimilate a whole new lexicon made up of software, listings, bugs, files, debugging, etc.

It is, however, a step that, sooner or later, must be taken by everyone, and therefore it must be taken now, since the time is more than ripe.

Programming

Programming machine movements in R3 language has many similarities with programming a PLC. In both cases the designer will have to:

- write a source program, that is, a list of operations to be performed by the machine, written in an appropriate language (R3 language in the Robox case). This is the design moment, i.e., the creative moment
- compile the source program using the appropriate compiler. The compiler, remember, translates the source program, written in a language close to human beings, into an object program, written in numerical codes understandable by the electronic controller that controls the machine
- load the object program into the electronic controller, an operation that is normally done using the serial or Ethernet communication ports
- check the operation of the machine and, if necessary, make changes to the source program

The fundamental difference between the R3 language and the ladder programming language is that the latter is designed for programming quantities of the attach/unattach

© 2025 Robox SpA

type, i.e., for example, tele-switches that open and close according to signals coming from pushbuttons, micro-switches, etc., which, in turn, can be opened or closed.

The R3 language, which is designed to regulate continuous quantities such as typically positions, velocities, torques, but also pressures, temperatures, etc., accepts and handles equations that correlate variables of this type.

There is practically no limit to the sophistication of operations: the R3 language handles integrals and derivatives, multiplications, divisions, sines, cosines, etc.

Evidently, the R3 language will also be able to handle switch on/off operations like a PLC, that is, to solve Boolean type equations, but this is certainly a secondary activity, compared to the main activity, which is aimed at controlling with precision, smoothness, rapidity, and obeying more or less complicated laws of motion, the aforementioned kinematic and non-kinematic quantities.

The extension of the R3 source file must be **.R3**

The extension of the inclusion files must be **.I3**

Programming with the R3 language is done in three different aspects.

Rules

A first programming step concerns the description of the laws of motion of the axes. For example, in the case of a motor that must rotate in perfect synchronism with a second motor (electric axis), the description of the law of motion will simply be:

```
IP2 = IP1
```

that is, the shaft position of the second motor must be identical to that of the first motor (and consequently the speeds will coincide with absolutely zero error).

If, conversely, two motors must have identical speeds minus a factor of K, the equation will be:

```
IV2 = IV1 * K
```

that is, the speed IV2 will have to coincide with the speed IV1 minus the K factor, and K can be any real number.

In the case then that a reel with a rocker is to be controlled, it will be written:

```
E = P - P0
INT_E = INT_E + E
IV1 = KP * E + KI * INT_E
```

Equation number 3. says that the rocker's error is the difference between the current position (i.e., P) and the centerline position (i.e., P0).

Equation number 5. says that the velocity of axis 1 must equal KP times the rocker's error (KP = proportional gain) + KI times the integral of the rocker's error (KI = integral gain).

Equations 1, 2, 3, 4, and 5 are referred to in R3 language as "rules" and will have to be checked with sufficiently high frequency over time, i.e., they will have to be honored continuously by a timed interrupt. The rules will not be fixed, but will vary according to the operational cycles of the machine.

Asynchronous rules

Rules invoked asynchronously are also managed: invocation can be done either by changes in the state of a digital input at a programmable frequency or by position pre-calculation.

The handling of these rules and the rules themselves are executed in such a way that they are not interrupted by any event. For the smooth operation of the system itself, the user must take great care in writing these rules so that their execution is as short as possible.

© 2025 Robox SpA

Operating cycles (main)

A second programming concerns the description of operational cycles and auxiliary functions.

For example, wanting to program that, if the Boolean variable "forward" is true, the electro-valve EVa will be energized or, if not, the EVi will be written:

```
...
if (forward)
out(EVa)
else
out(EVi)
endif
...
```

in this case, instructions should not be executed at a fixed rate, as in the case of rules, but whenever the program flow encounters them.

Categories of languages

The R3 language, as already mentioned, is a compiled language. The source program is translated by the compiler into an object program, and the object program, in turn, must be fed into the system memory.

The advantages of the compiled language are many; to make them clearer, let us recall that languages can be divided into three categories:

- interpreted languages
- compiled languages
- mixed languages

Interpreted languages

In interpreted languages, the various instructions, normally written in AscII format, are read and decoded at execution time. Execution will take place by calling appropriate routines in the memory (normally firmware) of the device.

The advantage of interpreted languages is greater ease of use, immediate executability, and easier debugging.

The disadvantage is the greater slowness of execution, the greater ease of making errors, and the lack of a "file" list.

Compiled languages

In compiled languages, the various instructions, still written in AscII format, are translated into object codes for the microprocessor that controls the controller and entered into its memory.

The advantages of compiled languages are among others:

- they allow maximum speed of execution of the program at the time of execution. This feature is extremely important in the R3 case for real-time solving of even very complicated equations

© 2025 Robox SpA

- they generate a file having the extension .LST (the list file) in which all formal errors are identified and documented and all useful information is given for subsequent program tuning
- ensure complete and exhaustive documentation of the program

Mixed languages

In mixed languages, the various instructions are translated into intermediate codes, which are not yet the object codes of the microprocessor that controls the controller, but are interpreted and executed by the resident software.

Mixed languages sometimes allow for the optimal trade-off, allowing speed of execution, ease of debugging, compactness of the generated object code.

Compilation

As mentioned earlier, the R3 language is a compiled language. The source program is translated by the compiler into hexadecimal codes executable by the microprocessor. The compiled version was preferred in order to give maximum priority to the speed of execution necessary to properly execute the instructions contained in the rules.

In the source file it is possible to introduce special instructions, called metacommands, which allow the compiler itself to vary, according to particular requirements.

The compilation generates, in addition to the object file, an .LST file containing general information, as well as a list of any syntactic grammatical errors contained in the source file. If these are major errors, the object file is not generated.

Robox Development Environment (RDE)

In order to facilitate the user in all these operations, a development environment called RDE has been developed. This development environment, which is usually hosted on Windows Personal Computers, though not being indispensable, creates the ideal environment for properly operating both the R3 language and the ROBOX Family modules.

Two types of programming

The R3 language provides for two types of programming.

The first type of programming is concerned with describing the laws of motion of the machine (rules programming).

The second type of programming, at the main level, concerns the description of the machine's operating sequences (instruction programming).

The rules should be executed at a frequency appropriate to the controlled process. By programming rules, only linear programming is permissible; no blocking loops or waiting for events should be used.

Clearly, the resources devoted by the controller to controlling machine movements (rules) will, in general, have to be much greater than those devoted to controlling operating sequences.

RULE-level programming

This level of programming is entrusted with the actual control of machine movements, in other words:

- the closing of position, speed or acceleration rings
- the control of any rockers
- the generation of appropriate trajectories

© 2025 Robox SpA

- the synchronization of multiple axes
- etc.

These functions must be described in the rules and must be verified by the "execution time" control with sufficiently high frequency compared to the bandwidth of the dynamics of the controlled process.

The modules of the Robox SpA family are designed to work with a rules update frequency ranging from 25 Hz to 2 KHz (default 100 Hz), which appears more than sufficient to control processes with bandwidths up to 200 Hz.

As mentioned, rule-level programming is given the task of describing the laws of motion of the machine.

In principle, a rule will relate to one of the *n* controlled axes, but it is permissible for a rule to describe the laws of motion of more than one axis. In order to be able to describe the laws of motion in the various operating states, it will be sufficient to write the different rules.

In general, three fields can be identified in a rule:

- REF
- MOTION
- AUX

In the REF field (optional) will be written the equations controlling the generation of the reference voltage to the servo drives (typically speed reference). If this field is omitted the control will use the standard rules.

In the MOTION field the laws of motion will be described.

In the AUX field will be described auxiliiary functions for which the same characteristic time requirements as for rules apply.

We repeat that the equations described in the rules are executed with a fixed frequency normally equal to 100 Hz, to which corresponds a scan period of 10 ms, with a maximum frequency of 2 KHz, to which corresponds a period of 0.5 ms. For each period, up to 32 rules can be executed.

MAIN-level programming

This level of programming is given the task of describing the operation of the machine.

Main-level programming is a classical type of programming, oriented toward the use of flowcharts. The user must define the flowchart(s) according to his needs and then convert them into R3 language instructions, an operation that normally presents neither conceptual nor practical difficulties.

To facilitate this, the R3 language allows multitasking main-level programming.

Up to 8 evolving programs can be described in parallel. All tasks have the same priority level, meaning that the available machine time will be allocated among all active tasks.

In order to facilitate the synchronization of the different tasks, an adequate number of global variables are available, i.e., accessible by the different tasks. Local variables can then be added to each task as needed.

When the control unit is switched on, only task #1 (autorun task) is automatically sent to execution. It will then be task 1 that will start the other tasks, if needed and if scheduled by the task 1 program.

While the program is running, it is also possible to disable one or more tasks by placing them in a waiting state. The system is also equipped with the ability to handle interrupts, i.e., external events at which the normal program must be dropped to allow such events to be handled with due speed.

© 2025 Robox SpA

The control of such an event is done within a RULE, and thus with a maximum latency time of nominally 20 microseconds. The rule in turn will force the execution of an appropriate user-written procedure at the main level.

Real time extended (RTE) inline operating system

The R3 language is supported in the Robox modules by an RTE real time operating system which provides among others the following basic functions:

- assigns the right partition of time to the evolution of rule-level programs
- assigns the right partition of time to the evolution of programs at the main level
- ensures the fundamental services at the axis control level
- ensures the fundamental services related to communications (RS232 RS422/485 ETHERNET)
- ensures the management (drivers) of a variety of I/O boards such as:
  - digital input output boards
  - analog input output boards
  - etc.
- ensures fundamental services at the diagnostic level
- enables the execution of asynchronous rules

Conclusions

To program in R3, the programmer will first have to assimilate its instructions. The instructions of the R3 language are the classic instructions of structured programming languages, with the addition of some specific instructions closely related to the problems for which the R3 language is intended.

Anyone familiar with programming languages such as Basic, C or others will take a few hours to assimilate the R3 language.

Anyone who is not familiar with the aforementioned languages but already has some experience in PLC programming will take a few days.

Finally, those with no experience whatsoever will be able to learn to program in a few weeks, provided they have a logical sense and are predisposed to mathematical sciences. Programming in R3 language then means knowing computers and being familiar with them.

The operator equipped with the above knowledge will sit down at a desk and begin to describe the application program for the machine of interest.

In this writing he will have at his disposal the so-called text editors, i.e., programs that allow text to be written, stored and corrected. The set of instructions written by the programmer will be called the source listing and will generally be easily understood by anyone. At this point the program will have to be compiled, that is, as already mentioned, translated into numerical codes understandable to the electronic controller.

This file will be called an object file. The electronic controller into which it is to be decanted will have to be connected to the personal computer via the RS232 or ETHERNET serial line with which every personal computer is equipped. The object program thus generated will be stored in the memories of the electronic controller.

The machine is ready to run, and the programmer will be able to verify that the behavior is as he intended. In general this does not succeed the first time: the operator will then have to check his program, find conceptual or formal errors, and correct them.

The R3 language and the RDE development environment returns this task particularly easily. Once the causes of the errors has been clarified, the developer will correct the source programs, recompile the program, and put it back into the unit memories.

© 2025 Robox SpA

Software requirements

Microsoft Windows 7 (x86), Windows 8 and 8.1 (x86), Windows 10 (x86/x64), Windows 11 (x86/x64) operating system.

Metacommands

Metacommands

Metacommands are identifiers intended to condition the compilation phase. They do not directly generate any code but specify how the R3 source program is to be processed.

Metacommands are divided into four groups:

- defining the programming mode general time diagram 770
- conditioning the listing 770
- general purpose 770
- pseudo instructions 771

Metacommands for defining the programming mode general time diagram

$rule	Definition of the Task 0 which contains the rules 720
$task n	Definition of the Task (1-:-8)
$rule_periodic	Definition of the periodic rule rule_periodic 730
$rule_time	Definition of the asynchronous rule int_timer 701, int_after 698
$rule_inp	Definition of the asynchronous rule int_inp 699

Metacommands for conditioning the listing

$define 772	Defines a symbol
$undef 774	Removes the definition of a symbol
$ifdef-$else-$endif 772	If the symbol is defined, then the program is compiled
$ifndef-$else-$endif 772	If the symbol is not defined, then the program is compiled
$expand	If present, the expansion of LITs within the list file is handled at compile time

Metacommands for general purposes

$include 773	Includes an external file in the compilation (*.i3). Useful for symbolic definition of hardware resources

© 2025 Robox SpA

$close_loop	Simulates the closure of the position loop. If present, RTE simulates the cp and cv values of axes
$check_array	Checks, run-time, if there is an access to a nonexistent array element. Obviously penalizes run-time
$write_on_extern	Enables ability to write variables defined as **extern** (public 787)
$use_rpe	Enables extension to RPE (Robox Path Executor) language. Automatically added using the RDE project configurator (axes groups)
$error "message"	From RC3E v33.01.21 Forces compiler to give a compilation error (useful, e.g., within a conditional block, to notice whether or not the block is being compiled)
$step	To obtain the max performance of the interrupting rules, unnecessary instructions haven't been included. If you want to re-enable the step storage in a asynchronous rule task (including RULE_PERIODIC) to set breakpoints on their execution, you have to add at the beginning of the file (after the file type definition) the line **$step** Example: **$rule_periodic** **$step**
$warning "messaggio"	From RC3E v33.1.21 Forces the compiler to give a compile warning (useful, e.g., within a conditional block, to notice whether or not the block is being compiled)

Pseudo instructions

Variables of type BIT_MASK allow the definition and initialization of particular integer variables (of type U32) to be used as bit masks.

Typically the setting of bits is done automatically through the configuration choices the user makes in the RDE project configurator.

However, variables of type BIT_MASK can also be read and modified from source file in R3.

bit_mask_set namevar value	The variable namevar is initialized to value
bit_mask_add namevar value	In namevar the bits specified by value are set to 1, without altering the others
bit_mask_remove namevar value	In namevar the bits specified by value are set to 0, without altering the others

© 2025 Robox SpA

NOTE: Even the lit 702 instruction allows you to define symbolic to the variables.

Metacommand $define

It allows you to define a symbol that will later be used to condition the compilation ($ifdef 772, $ifndef 772).

Syntax	**$define symbol** **$define symbol = value**
symbol	The name of the symbol to define
value	The value to be associated with the symbol

Examples

Example of use:

```
; I define the version symbol and assign it the value 12
$define version = 12
; I define the symbol test_version
$define test_version
; if version is greater than or equal to 12 and test_version is
false...
$ifdef 772 version >= 12 and not test_version
 ; then this part of the code will be compiled
 ...
$endif 772
```

Metacommand $ifdef/$ifndef

Executes the if condition based on whether or not a symbol is defined.

Syntax	**$ifdef bool exp1 [op] [bool expN]]** ... **$else** ... **$endif**
exp1	These is a boolean expression composed as follows: • symbol <opr> const where <opr> is an operator chosen from: >, >=, <, <=, ==, <> and const is a numeric constant
op	Logic operator 782: and, and not, or, or not, xor (optional)

© 2025 Robox SpA

Syntax	**$ifdef bool exp1 [op] [bool expN]]** ... **$else** ... **$endif**
expN	These is another boolean expression (max. 5 included *exp1*) (optional)
Note	Priority rules: *exp1*,.., *expN* is evaluated first, then the sequence of other expression from left to right is evaluated, with no priority applied, between the various *op*

Examples

Example of use:

```
; I define the version symbol and assign it the value 12
$define 772 version = 12
; I define the test_version symbol
$define 772 test_version
; if version is greater than or equal to 12 and test_version is
false...
$ifdef version >= 12 and not test_version
    ; then this part of the code will be compiled
    ...
$else
    ; otherwise this part of the code will be compiled
    ...
$endif
```

Metacommand $include

Allows the indicated external file to be incorporated into the compilation.

Syntax	**$include filename**
filename	Name of the file to include. The file refers to the folder of the file containing the $include directive, unless it is specified with absolute path
Note	Multiple files can be included (max. 400)

Examples

Example of use:

```
; In the main (main program)
```

© 2025 Robox SpA

```
$task 1
; definition of the input/output
$include litio.i3
; definition of registers
$include litreg.i3
```

Example of the contents of the file litio.i3:

```
...
lit iAirComp          inp(145)
lit iManual           inp(146)
...
```

Example of the contents of the file litreg.i3:

```
...
lit rrVelMan          nvrr(1)
lit rrAccMan          nvrr(2)
...
```

Metacommand $undef

Allows you to remove the definition of a symbol previously defined using $define 772.

Syntax	**$undef symbol**
symbol	Name of the symbol to be removed

Examples

Example of use:

```
$define 772 PROX_ABRASIVE
$define 772 BRIDGE_2

...

$undef PROX_ABRASIVE
$undef BRIDGE_2
```

Operators

Operators

Operators are those characters, or keywords, that allow variables, constants and functions to be related to generate more or less complex expressions.

Operators can be divided into six groups:

- assignment operator 781 (=)
- arithmetical operators 776 (+ , - , * , /)
- logical operators 782 (**and**, **or**, **not**, **xor**)

- numerical relational operators 784 (> , < , = , <> , >= , <=)
- binary operators on integer variables 777 (**r_and**, **r_not** , **r_or** , **r_xor**)
- string concatenation operator 781 (#)

The natural precedences of operator execution are given in the table below. In general, the precedences are in the order:

- Unary
- Arithmetical
- Relationalal
- Binary on bits
- Binary logic
- Assignment

Brackets should be used to impose the desired order of execution. In general, it is advisable to add pairs of brackets whenever expressions may raise doubts about precedence.

Precedence is according to the order of writing. Groups identify the same level of precedence.

Minus unary	-	- var
Plus unary	+	+ var
Not logical	**not**	**not** var
Not binary	**r_not**	**r_not** (var)
Multiplication	*	var * var
Division	/	var / var
Addition	+	var + var
Subtraction	-	var - var
Higher than	>	var > var
Higher than or equal	> =	var > = var
Less than	<	var < var
Less than or equal	< =	var < = var
Equal	=	var = var
Different	< >	var < > var
And binary	**r_and**	var **r_and** var
Or binary	**r_or**	var **r_or** var
Xor binary	**r_xor**	var **r_xor** var
And logical	**and**	var **and** var
Or logical	**or**	var **or** var

© 2025 Robox SpA

Assignment	=	var = var

Arithmetical operators

Validity: **Rule, Task**

In general, these operators work between integer or real quantities.

If a real quantity is assigned to a variable of integer type, the decimal digits are truncated.

If the value is too large to be represented in the integer variable, the result will be unpredictable.

When an arithmetical operator combines a real with an integer, the result is converted to a real.

Addition

Sum operator between two numerical quantities.

Syntax	**valueA + valueB**
valueA	First addend value
valueB	Second addend value
Result	The sum of the two values

Examples:

```
; The register rr(12) will contain the sum of the current position of
axis 2 cp(2)
; and the contents of register rr(3)
rr(12) = cp(2) + rr(3)
; It can also be used as a sign (unary) operator.
; The register r(12) will contain the value +3
r(12) = +3
```

NOTE: To sum (concatenate) strings, use the operator # 781.

Subtraction

Subtraction operator between numerical quantities.

Syntax	**valueA - valueB**
valueA	Value of the minuend
valueB	Value of the subtrahend
Result	The difference between minuend and subtrahend

Examples:

```
; The register rr(12) will contain the difference between the contents
of the register r(2)
; and the contents of the variable remainder
```

© 2025 Robox SpA

```
rr(12) = r(2) - remainder
; It can also be used as a sign (unary) operator.
; The register r(12) will contain the value -3
r(12) = -3
```

Multiplication

Multiplication operator between numerical quantities.

Syntax	**valueA * valueB**
valueA	Value of the first factor
valueB	Value of the second factor
Result	The multiplication between the factors

Examples:

```
; The register rr(12) will contain the product between the constant K_PI
(Pi)
; and the variable diameter
rr(12) = K_PI * diameter
```

Division

Division operator between numerical quantities.

Syntax	**valueA / valueB**
valueA	Dividend value
valueB	Value of the divisor (different from 0)
Result	The division between dividend and divisor

NOTE: In the case of division between integers, the result is only the integer part of the division even if the destination is a real quantity.

Examples:

```
; The register rr(12) will contain the quotient between the value 2.0
and the
; variable "value" in turn divided by the variable "remainder"
REAL value = 10.0
REAL remainder = 0.5
rr(12) = (2.0 / value) / remainder
```

Binary operators (bit)

Validity: **Rule**, **Task**

The use of this family of operators is reserved for integer values. If used with real values, the compiler will issue an error. The result is still a value of type integer.

AND binary

© 2025 Robox SpA

Bit-to-bit (binary) AND operation between two integer values.

Syntax	**valueA r_and valueB**
valueA	First value
valueB	Second value
Result	Binary AND between the two values

Examples:

```
; With the data given in the table, value1 will contain the value 0xC0
value1 = inp_w(1) r_and 0c8h

; Binary result of the operation
```

	B3								B2								B1								B0							
inp_w(1)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	1	1	0	1	0	1	0	0
0c8h	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0
value1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0

```
; value2 will contain the value of the lowest byte of r(1)
value2 = r(1) r_and 0xFF
```

OR binary

Bit-to-bit (binary) OR operation between two integer values.

Syntax	**valueA r_or valueB**
valueA	First value
valueB	Second value
Result	Binary OR between the two values

Examples:

```
; With the data given in the table, value1 will contain the value 0xF6DC
value1 = inp_w(1) r_or 0c8h
```

```
; Binary result of the operation
```

	B3								B2								B1								B0							
inp_w(1)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	1	1	0	1	0	1	0	0
0c8h	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0
value1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	1	1	0	1	1	1	0	0

NOT binary

NOT bit-to-bit (binary) operation for an integer value.

Syntax	**r_not value**
value	Value
Result	Binary NOT of value

Examples:

```
; With the data given in the table, value1 will contain the value
0xFFFF092A
value1 = r_not inp_w(1)

; Binary result of the operation
```

	B3								B2								B1								B0							
inp_w(1)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	1	1	0	1	0	1	0	0

© 2025 Robox SpA

value1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	1	0	0	1	0	0	1	0	1	0	1	0

XOR binary

Bit-to-bit (binary) XOR operation between two integer values.

Syntax	**valueA r_xor valueB**
valueA	First value
valueB	Second value
Result	Binary XOR between the two values

Examples:

```
; With the data given in the table, pippo will contain the value 0xF61C
pippo = inp_w(1) r_xor 0C8H

; Binary result of the operation
```

	B3								B2								B1								B0							
inp_w(1)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	1	1	0	1	0	1	0	0
0c8h	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0
value1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	0	0	0	1	1	1	0	0

Assignment operators

Validity: **Rule, Task**

Allows a variable to be assigned the value of a constant, either a variable or an expression.

Syntax	**var = valueS** **var = varS**
var	Destination variable
valueS	Constant value or source expression
varS	Source variable

NOTE: Assigning to an integer variable a real value, it will result in truncation of the decimal part; if the value is too large to be represented in the integer variable, the result will be unpredictable.

Examples:

```
; The register r(100) will contain the value of the product 17*3
; minus the value of the variable value1
r(100) = 17 * 3 - value1
; The variable value2 will take the value of the contents of the
register r(100)
; multiplied by the constant K_PI (Pi).
value2 = r(100) * K_PI
; The string register sr(12) will contain the string:
; This is a string..
sr(12) = "This is a string.."
```

String concatenation operators

Validity: **Rule, Task**

String concatenation operator, can be used only with variables of type string.

Syntax	**valueA # valueB**
valueA	First string
valueB	Second string
Result	The concatenation between the two strings

Examples:

```
; In both cases the string register sr(1) will contain the string: "it
is eight o'clock"
sr(1) = "it is" # " eight o'clock"
sr(1) = "it is" # " " # "eight o'clock"
```

© 2025 Robox SpA

Logical operators

Validity: **Rule, Task**

These operators operate between binary boolean quantities and returns a result of binary boolean type, that is, a result that can have only two values: "true" or "false".

AND logical

Logical conjunction between boolean expressions.

Syntax	**valueA and valueB**
valueA	First boolean value
valueB	Second boolean value
Result	

valueA	valueB	Result
false	false	false
true	false	false
false	true	false
true	true	true

Examples:

```
; If the contents of register rr(12) is zero and input(5) is active
(i.e., non-zero)
; then the string register sr(1) will be assigned the string:
; "the variable rr(12) is equal to zero and input 5 is TRUE"
_if ((rr(12) = 0) and inp(5))
    sr(1) = "the variable rr(12) is equal to zero and input 5 is TRUE"
; If the contents of registers r(12) and r(5) are non-zero
; then the string register sr(1) will be assigned the string:
; "both registers r(12) and r(5) are different from zero."
_if (r(12) and r(5))
    sr(1) = "Both register r(12) and r(5) are different from zero"
```

NOT logical

Logical negation operator of a boolean expression.

Syntax	**not value**
value	Boolean value
Result	

value	Result
false	true
true	false

Examples:

```
; If it is false that the value of the variable "value1"
```

```
; is equal to the contents of register r(5)
; then the string register sr(3) will be assigned the string:
; "the variable value1 is different from the contents of r(5)"
if (not (value1 = r(5))
 sr(3)= "the variable value1 is different from the contents of r(5)"
endif
```

OR logical

Logical disjunction between boolean expressions.

Syntax	**valueA or valueB**
valueA	First boolean value
valueB	Second boolean value
Result	

valueA	valueB	Result
false	false	false
true	false	true
false	true	true
true	true	true

Examples:

```
; if the value of the variable "value1" is non-zero
; or input(5) is active (i.e. different from zero)
; then the string register sr(3) will be assigned the string:
; "variable value1 is <> 0 or input 5 is TRUE"
if (value1 or inp(5))
 sr(3) = "the variable value1 is <> 0 or input 5 is TRUE"
endif
```

XOR logical

Exclusive logical disjunction between boolean expressions.

Syntax	**valueA xor valueB**
valueA	First boolean value
valueB	Second boolean value
Result	

valueA	valueB	result
false	false	false
true	false	true
false	true	true
true	true	false

© 2025 Robox SpA

Examples:

```
; If inp(1) and inp(2) are both true or both false
; then the boolean variable returns will be assigned the value false
; if one of the two inputs is true and the other is false
; then the boolean variable returns will be assigned the value true
returns = xor(inp(1), inp(2))

; if only one of the two inputs is true and the other is false
; then the contents of register r(1) will be incremented by one unit
if (xor(inp(1), inp(2)))
 inc(r(1) ,1)
endif
```

Numerical relational operators

Validity: **Rule**, **Task**

These operators operate on numerical quantities. Use on quantities of logical type should be avoided while comparing strings requires the use of the appropriate functions.

Minor operator

Syntax	**valueA < valueB**
valueA	First value
valueB	Second value
Result	True if valueA less than valueB, false otherwise

Examples:

```
; If the contents of register r(2) is less than the contents of register
rr(3)
; then the following "if" instructions will be executed
if (r(2) < rr(3))
    ...
endif

; If the condition is false (i.e., the contents of register nvr(1)
; is not less than the contents of register nvr(2))
; then "return" will be assigned the value 0 (return = 0).
; If the condition is true (i.e., the contents of the register nvr(1)
; is less than the contents of register nvr(2))
; then "return" will be assigned the value 1 (return = 1)
return = (nvr(1) < nvr(2))
```

Operator less than or equal to

Syntax	**valueA <= valueB**
valueA	First value

© 2025 Robox SpA

Syntax	**valueA <= valueB**
valueB	Second value
Result	True if valueA less than or equal to valueB, false otherwise

Examples:

```
; if the variable "value1" is less than or equal to 3.1415
; then the following "if" statements will be executed
if (value1 <= 3.1415)
 ...
endif

; If the condition is false (that is, the contents of register nvr(1)
; is not less than or equal to the contents of register nvr(2))
; then "return" will be assigned the value 0 (return = 0).
; If the condition is true (i.e., the contents of register nvr(1)
; is less than or equal to the contents of register nvr(2))
; then "return" will be assigned the value 1 (return = 1)
return =(nvr(1) <= nvr(2))
```

Major operator

Syntax	**valueA > valueB**
valueA	First value
valueB	Second value
Result	True if valueA greater than valueB, false otherwise

Examples:

```
; if the contents of register rr(2) is greater than the contents of
register r(3)
; then the following "if" statements will be executed
if (rr(2) > r(3))
 ...
endif

; If the condition is false (that is, the contents of register nvr(1)
; is not greater than the contents of register nvr(2))
; then "return" will be assigned the value 0 (return = 0).
; If the condition is true (i.e., the contents of register nvr(1)
; is greater than the contents of register nvr(2))
; then "return" will be assigned the value 1 (return = 1)
return = (nvr(1) > nvr(2))
```

Operator greater than or equal to

© 2025 Robox SpA

Syntax	**valueA >= valueB**
valueA	First value
valueB	Second value
Result	True if valueA greater than or equal to valueB, false otherwise

Examples:

```
; If the variable "value1" is greater than or equal to 3.1415
; then the following "if" statements will be executed
if 696 (value1 >= 3.1415)
 ...
endif 683

; If the condition is false (that is, the contents of register nvr(1)
; is not greater than or equal to the contents of register nvr(2))
; then "return" will be assigned the value 0 (return = 0).
; If the condition is true (that is, the contents of register nvr(1)
; is greater than or equal to the contents of register nvr(2))
; then "return" will be assigned the value 1 (return = 1)
return =(nvr(1) >= nvr(2))
```

Equality operator

Syntax	**valueA = valueB**
valueA	First value
valueB	Second value
Result	True if valueA equals valueB, false otherwise

Examples:

```
; If the contents of register r(2) is equal to the variable "value1"
; then the following "if" statements will be executed
if (r(2) = value1)
 ...
endif

; If the contents of register rr(2) is equal to the variable "value2"
using
; the similar() function, then the following instructions at the "if"
will be executed
if(similar 575(rr(2), value1)
 ...
endif

; If the condition is false (that is, the contents of register nvr(1)
; is not equal to the contents of register nvr(2))
; then "return" will be assigned the value 0 (return = 0).
; If the condition is true (i.e., the contents of register nvr(1)
```

© 2025 Robox SpA

```
; is equal to the contents of register nvr(2))
; then "return" will be assigned the value 1 (return = 1)
return = (nvr(1) = nvr(2))
```

NOTE: We do not recommend using the = operator to compare real quantities (or a real with an integer): instead, we recommend using the appropriate similar() function.

Not equality operator

Syntax	**valueA <> valueB**
valueA	First value
valueB	Second value
Result	True if valueA different from valueB, false otherwise

Examples:

```
; if the variable "value" is different from 3.1415
; then the following "if" statements will be executed.
if (value1 <> 3.1415 )
 ...
endif

; If the condition is false (that is, the contents of register nvr(1)
; is not different from the contents of register nvr(2))
; then "return" will be assigned the value 0 (return = 0).
; If the condition is true (i.e., the contents of register nvr(1)
; is different from the contents of register nvr(2))
; then "return" will be assigned the value 1 (return = 1)
return = (nvr(1) <> nvr(2))
```

Data types

Variables are entities whose value is not known during the writing of a program and normally varies during its execution.

The data types used by the R3 language are:

Type	Description
I8	8-bit signed integer variable
I16	16-bit signed integer variable
I32	32-bit signed integer variable
U8	8-bit unsigned integer variable
U16	16-bit unsigned integer variable
U32	32-bit unsigned integer variable
INT	32-bit integer variable (it is like I32)

© 2025 Robox SpA

Type	Description
FLOAT 892	Floating point variable 4 Byte
REAL 898	Floating point variable 8 Byte (double)
BOOL 890	Boolean variable
CHAR 891	Char type variable (it is like I8)
STRUCT 888	Structure type variable
STRUCT_P 889	Packed structure type variable (data is contiguous, takes up less space)
STRING 899	String type variable (127 char + terminator)
TABSTR 889	Tables of string constants
COUNTER 892	Variable of type counter
TIMER 900	Variable of type timer
POWER_SET 896	Variable for power handling
BCC_MSGUSER 793	Support structure for functions bcc_send() 206 , bcc_receive() 204, ...
STRU_BCC_TCP_CLIENT 803	Support structure for functions bcc_get_tcp_clients_info() 200 and getTcpClientsInfo()
STRU_AAX 795	Support structure for functions planar_comp_ip 494() and planar_comp_cp 493 ()
STRU_AXIO_PDI 801	Support structure for functions axio_robj() 162 and axio_wobj() 177
STRU_CO_OBJ 805	Support structure for functions co_robj() 234 and co_wobj() 249
STRU_COE_OBJ 807	Support structure for functions co_robj 234 () and co_wobj() 249
STRU_COERRMSG 809	Support structure for function co_rerr 231()
STRU_STAT 882	Support structure for function statistics() 595
STRU_GROR 841	Support structure for function group 694 order() 708
STRU_FILEINFO 816	Support structure for function f_dir() 297
STRU_MVITP 850	Support structure for movement function mva_to_n() 471
STRU_MATCAM 846	Support structure for movement function mv_cam() 425
STRU_MODBUS 847	Support structure for communication functions modbus/tcp 116

© 2025 Robox SpA

Type	Description
STRU_CAM 804	Support structure for movement function mv_table() 455
STRU_MVTO 853	Support structure for movement function mv_to() 457
STRU_MVTOCJ 855	Support structure for movement function mv_to_cj() 461
STRU_MVTOCJV 857	Support structure for movement function mv_to_cjv() 461
STRU_MVTOCJV_INFO 859	Support structure for function mv_to_cjv_info() 463
STRU_MVTOCJVE 861	Support structure for movement function mv_to_cjv() 461
STRU_MVZC 863	Support structure for movement function mva_zc() 486
STRU_MVGENITP 849	Support structure for movement function mva_to_n_v() 479
STRU_MVITPCJ 852	Support structure for movement function mva_to_n_cj() 475
STRU_AH_D 796	Support structure for function ah_get_d() 136
STRU_AH_D_EXT 797	Support structure for function ah_get_d() 136 with USV
STRU_AH_INFO 798	Support structure for function ah_info() 139
STRU_AS_D 799	Support structure for function as_get_d() 154
STRU_AS_D_EXT 800	Support structure for function as_get_d() 154 with USV
STRU_FILEHND 815	Support structure for functions f_open() 302, f_close() 294, ..
STRU_INT 842	Support structure for functions int_after() 698, int_inp() 699 e int_timer() 701
STRU_LARGE_INT 844	Support structure for functions get_i64() 692 e set_i64() 742
STRU_LARGE_UINT 844	Support structure for functions get_u64() 693 e set_u64() 747
STRU_CRIMPER 812	Support structure for movement function mv_crimper() 427
STRU_ECAT_PROBE 814	Support structure for function ecat_probe() 288
STRU_FOLLOW 816	Support structure for movement function mv_follow() 428

© 2025 Robox SpA

Type	Description
STRU_FOLLOW2 432	Support structure for movement function mv_follow2() 432
STRU_FOLLOW2_S 432	Support structure for movement function mva_follow2() 432
STRU_FOLLOW2_SM 432	Support structure for movement function mva_follow2_m() 432
STRU_FOLLOW2_INFO 827	Support structure for movement function mv_follow2() 432
STRU_HW_INFO 841	Support structure for function get_hw_info() 326
STRU_PHADJ 868	Support structure for movement function mv_phase_adj() 447
STRU_PHADJ2 871	Support structure for movement function mv_phase_adj2() 448
STRU_PM_PD 875	Support structure for function axio_pm_r() 160
STRU_REACHTARGET 876	Support structure for movement function mv_reach_target() 451
STRU_SAFETY_PRJINFO 878	Support structure for function safe_prj_info() 549
STRU_SOE_IDN 879	Support structure for functions soe_ridn() 577 e soe_widn() 589
STRU_SYNCHRO 884	Support structure for movement function mv_synchro() 454
STRU_TRACK 886	Support structure for movement function tracking() 622
STRU_MASTERF 845	Support structure for function master_filter() 364

NOTE: Nesting of structures (structures of structures) is allowed.

Variables can have the following levels of visibility:

Visibility	Description
Global	These are the R3 language's predefined variables or locales declared as public (see below). They are accessible from all tasks and usable on all debugging tools
Local	Accessible variables (R/W) from the task in which they are declared only. They must be defined at the beginning of the source file before the instructions. `INT pluto`

Visibility	Description
Public	Variables accessible (R/W) by the task to which they are declared and normally readable by other tasks. (see external) They must be defined at the beginning of the source file, before instructions, prefixed with "PUBLIC". `PUBLIC INT pluto`
External	Variables from a different task. They are normally read-only unless the task that includes them has the metacommand "$WRITE_ON_EXTERN 770" at its beginning. They must be defined at the beginning of the source file with the prefix "EXTERN" to which the equivalent PUBLIC must necessarily correspond in another task. (see Public) `EXTERN INT pluto`

Retentive	User structure of which retentiveness is desired at shutdown.(retentive memory)
	
	It has the following special features:
	
	• Maximum size of the structure is smaller than that of the user area of retentive memory (see report)
	• Cannot be an array of structure (but nothing prohibits it from being inside it)
	• Maximum size of structure name of 31 characters
	• Maximum variable name size of 31 characters
	• The metacommand $ WRITE_ON_EXTERN 787 must be present in order to write this structure.
	• The association of the user structure to the retentive structure occurs only at load time if and only if NO previous associations exist
	• If a retentive structure already exists then, at task startup, the structure name, instance name, and size are checked to match those previously recorded
	• This structure is saved, as are the registers, when the control is turned off
	
	Example:
	
	`$WRITE_ON_EXTERN`
	`STRUCT ret_struct`
	`  INT integerData[20];`
	`  REAL realData[10];`

© 2025 Robox SpA

Visibility	Description

	`STRING setName[64];`
	`END_STRUCT`
	`RETENTIVE ret_struct retentiveData`
	`retentiveData.integerData[0] = SECOND`
	`retentiveData.integerData[1] = MINUTE`
	`retentiveData.integerData[2] = HOUR`
	
	NOTE: To delete the association of the user structure to the retentive structure use the device command (ascii directive) REMOVE NV_USERSTRUCT.
	
	NOTE: To initialize the structure in case of memory loss use the device command (ascii directive) RESTORE NV_USERSTRUCT.
	
	NOTE: To automatically enlarge the size of the structure, provided the structure and instance names are the same, use bit 10 of the default variable SYS_CFG.

Local to functions	Variables that are defined within function() 688 and are usable only within it. These variables lose their contents each time the function exits. **NOTE**: It is possible to redefine the name of local variables, that means, internal variables can have the same name as local variables. However, it is recommended to use different names to avoid confusion. **WARNING**: These are different locations, that means, internal and local variables with the same name occupy different physical memory locations. Changing an internal variable has no effect on a local variable that has the same name. Several variables with the same name can be declared in functions. They are NOT usable by debugging tools. They can also be placed in motion 703 fields within rules 720. They behave similarly to those declared in functions.

It is possible to declare local variables in rules, as long as they are within the REF / MOTION / AUX blocks. Variables in the REF block are visible only in the block itself; those in MOTION and AUX are in common with each other.

You can declare local variables within the task main code (visibility is limited between the point of declaration and the end of main).

Examples

Various examples of data declaration:

```
; "varname" is a character string that must necessarily begin with a
letter.
; variable definition has the following structure:
; single variable definition of type 'typeData'
typeData varname
; definition of n variables of type 'typeData'
typeData varname1, varname2, ..., varnameN
; definition of a mono-dimensional array of n elements of type
'typeData'
; element indices range from 0 to n-1, so:
; the first element is varname[0] and the last element is varname[n-
1]
typeData varname[n]
; definition of a two-dimensional array of n*m elements of type
'typeData'
; element indices range from 0 to n-1 and 0 to m-1, so:
; the first element is varname[0][0] and the last element is
varname[n-1][m-1]
typeData varname[n][m]
; definition of a three-dimensional array of n*m*o elements of type
'typeData'
; element indices range from 0 to n-1, 0 to m-1, and 0 to o-1, so:
; the first element is varname[0][0] and the last element is
varname[n-1][m-1][o-1]
typeData varname[n][m][o]
; single initialized variable definition of type 'typeData'
; i.e., the variable is assigned the value k
typeData varname = k
; definition of an initialized array of type 'typeData'
; the individual elements are assigned the values k1, k2,..., kn in
the following manner:
; varname[0]=k1, varname[1]=k2, ... , varname[n-1]=kn,
typeData varname[n] = {k1, k2,...,kn}
; definition of a variable that is also visible for reading by the
other tasks.
; which declare it as extern (use the $WRITE_ON_EXTERN 787 metacommand
to be able to write it as well)
public typeData varname
; definition of a read-only variable declared in another task
extern typeData varname
```

Structure BCC_MSGUSER

Validity: **RC3E v33.3.1** (or later)

This keyword allows a variable of type BCC_MSGUSER (STRUCT_P 889) to be defined.

This variable is used by the send and receive message instructions bcc_send 206() and bcc_receive 204().

It consists of the following information:

Field	Type	Description
resSys1	U32	Reserved
resSys2	U8	Reserved
bccID	U8	BCC protocol identifier
dst	U8	Destination
dch	U8	Destination channel ID
src	U8	Source ID
sch	U8	Source channel ID
msg	U16	Message Code
lendata	U8	Message data size in bytes
pid	U8	Protocol ID
dlc	U8	Dead loop counter
data[255]	STRING	Data buffer

Examples

Example of declaration:

```
; definition of the variable mymessage of type BCC_MSGUSER
BCC_MSGUSER mymessage
```

Example of use:

```
; use of data contained in a field of the structure
; definition of mymessage variable as BCC_MSGUSER type
BCC_MSGUSER mymessage
; define a structure "MIA_STRUCT" type STRUCT_P 889
STRUCT_P mia_struct
    I32 d1
    I32 d2
    REAL d3
END_STRUCT
; define the variable my_data type MIA_STRUCT
mia_struct my_data
; transfer the contents of the "data" field of the miomessaggio
structure
; in the my_data structure
move_byte 704 (miomessaggio.data, my_data, SIZEOF(my_data))
; execute the operations on the data read
if(my_data.d1 = 27)
    ...
endif
```

© 2025 Robox SpA

Structure STRU_AAX

This keyword allows you to define a variable of type STRU_AAX (STRUCT_P 889). This structure contains the positions of the compensated axes.

It can be used in the planar_comp_ip 494() and planar_comp_cp 493() functions.

It consists of the following information:

Field	Type	Description
axp [1-:-32]	REAL	Array for axis positions

Examples

Example of use:

```
; SAMPLE CODE
; Global Storage definition
;---------------------------
STRU_aax in_cp, out_cp
STRU_aax in_ip, out_ip

in_ip.axp[1] = IP(1)
in_ip.axp[2] = IP(2)
in_ip.axp[3] = IP(3)
; When the compiler is ready you write:
; esito = PLANAR_COMP_IP (in_ip, out_ip);
; until the compiler it's not ready... using the right files:
; C:\Robox\Rc3e\33.5.16\include\parnames.def
; C:\Robox\Rc3e\33.5.16\include\prdf_str.inr
; C:\Robox\Rc3e\33.5.16\include\struser.h
; C:\Robox\Rc3e\33.5.16\cpu-ppc-p2020\include\rrt\interrt.h
; C:\Robox\Rc3e\33.5.16\cpu-ppc-g2\include\rrt\interrt.h
; you can use the function in the following way
$LIN_STEP planar_comp_ip 494(&in_ip, &out_ip)
rr(120+1) = out_ip.axp[1]
rr(120+2) = out_ip.axp[2]
rr(120+3) = out_ip.axp[3]

in_cp.axp[1] = CP(1)
in_cp.axp[2] = CP(2)
in_cp.axp[3] = CP(3)
; when the compiler is ready you write:
; esito = PLANAR_COMP_CP (in_cp, out_cp);
$LIN_STEP planar_comp_cp 493(&in_cp, &out_cp)
rr(160+1) = out_cp.axp[1]
rr(160+2) = out_cp.axp[2]
rr(160+3) = out_cp.axp[3]
```

© 2025 Robox SpA

Structure STRU_AH_D

This keyword allows a variable of type STRU_AH_D (STRUCT_P 889) to be defined.

It is used by the ah_get_d 136() alarm handling instruction.

It consists of the following information:

Field	Type	Description
Id	I32	Alarm identifier
hour	I32	Hour in which the alarm occurred
min	I32	Minute in which the alarm occurred
sec	I32	Second in which the alarm occurred
day	I32	Day in which the alarm occurred
mon	I32	Month in which the alarm occurred
year	I32	Year in which the alarm occurred
alnum	I32	Number of the alarm
alaxis	I32	Axis to which the alarm refers
p1	REAL	Possible first parameter of the alarm
p2	REAL	Possible second parameter of the alarm
p3	REAL	Possible third parameter of the alarm

Examples

Example declaration:

```
; definition of variable struAhD of type STRU_AH_D
STRU_AH_D struAhD
```

Example of use:

```
; use of the data contained in a field of the structure
; definition of struAhD variable of type STRU_AH_D
STRU_AH_D struAhD
; I define the variables
I32 alarm_number
I32 returned
I32 alarm_index = 2
```

© 2025 Robox SpA

```
; I read an alarm from the alarm history
returned = ah_get_d(index_all, struAhD)
; I read the value of the field "alarm number" alnum
; and use it as the selection variable of a SELECT block ...
CASE
alarm_number = struAhD.alnum
select(alarm_number)
 case 1
 sr(1) = "warning! MAJOR ALARM"
 break
 case 2
 sr(1) = "Non-blocking alarm"
 break
 case 3
 sr(1) = "Malfunctioning"
 break
 default
 sr(1) = "No alarm"
end_select
```

Structure STRU_AH_D_EXT

This keyword allows a variable of type STRU_AH_D_EXT (STRUCT_P) to be defined.

It is used by the ah_get_d() alarm handling instruction.

It consists of the following information:

Field	Type	Description
Id	I32	Alarm identifier
hour	I32	Hour in which the alarm occurred
min	I32	Minute in which the alarm occurred
sec	I32	Second in which the alarm occurred
day	I32	Day in which the alarm occurred
mon	I32	Month in which the alarm occurred
year	I32	Year in which the alarm occurred
alnum	I32	Number of the alarm
alaxis	I32	Axis to which the alarm refers

© 2025 Robox SpA

Field	Type	Description
p1	REAL	Possible first parameter of the alarm
p2	REAL	Possible second parameter of the alarm
p3	REAL	Possible third parameter of the alarm
us1	STRING[64]	First user string variable USV
us2	STRING[64]	Second user string variable USV
us3	STRING[64]	Third user string variable USV

Examples

Example of use:

```
stru_ah_info ahi
stru_ah_d_ext ah
int result
int ahidcnt
...
; get alarm history information
ah_info(ahi);
ahidcnt = ahi.oldid
while (ahidcnt <= ahi.newcnt)
    result = ah_get_d(ahIdcnt, ah)
    inc (ahidcnt)
    ...
    ...
end_while
...
```

Structure STRU_AH_INFO

This keyword allows a variable of type STRU_AH_INFO (STRUCT_P 889) to be defined.

It is used by the ah_info 139() alarm management instruction.

It consists of the following information:

Field	Type	Description
oldId	I32	Oldest progressive number of alarms
newId	I32	Newest sequence number of alarms
nal	I32	Alarm number

© 2025 Robox SpA

Field	Type	Description
ahSize	I32	Alarm size

Examples

Example of use:

```
BOOL get_ah_info
INT result_ah_info
STRU_AH_INFO info_alarm_histrory
STRING rep_msg

; ...

if(get_ah_info)

    get_ah_info = false
    result_ah_info = ah_info 139(info_alarm_histrory)

    if(not result_ah_info)
        inf_report 697(0x20, "Alarm histrory info:")
        str_format 601(rep_msg, "AH old id: %i",
info_alarm_histrory.oldId)
        inf_report(0x20, rep_msg)
        str_format(rep_msg, "AH new id: %i",
info_alarm_histrory.newId)
        inf_report(0x20, rep_msg)
        str_format(rep_msg, "AH num al: %i", info_alarm_histrory.nal)
        inf_report(0x20, rep_msg)
        str_format(rep_msg, "AH size: %i",
info_alarm_histrory.ahSize)
        inf_report(0x20, rep_msg)
    else
        inf_report(0x80, "### Alarm history not enabled! ###")
    endif
endif
```

Structure STRU_AS_D

Validity: **RC3E v33.3.4** (or later)

This keyword allows a variable of type STRU_AS_D to be defined.

It is used by the alarm handling instruction as_get_d 154().

It consists of the following information:

Field	Type	Description
alnum	I32	Alarm number
alaxis	I32	Axis that generated the alarm

© 2025 Robox SpA

Field	Type	Description
p1	REAL	First parameter of the alarm
p2	REAL	Second parameter of the alarm
p3	REAL	Third parameter of the alarm

Examples

Example of use:

```
; definition of variable s_al of type STRU_AS_D
STRU_AS_D s_al

; I read the information about the alarm, if any, in the stack
as_get_d(1, s_al)

; I perform operations on the read data
; I check whether there is alarm 2000 at the position read with the
first parameter equal to 23 ...
if ((s_al.alnum = 2000) and (s_al.p1 = 23))
    ...
endif
```

Structure STRU_AS_D_EXT

Validity: **RTE v34.30.0** (or later)

This keyword allows a variable of type STRU_AS_D_EXT to be defined.

It is used by the alarm handling instruction as_get_d().

It consists of the following information:

Field	Type	Description
alnum	I32	Alarm number
alaxis	I32	Axis that generated the alarm
p1	REAL	First parameter of the alarm
p2	REAL	Second parameter of the alarm
p3	REAL	Third parameter of the alarm
us1	STRING[64]	First user string variable USV
us2	STRING[64]	Second user string variable USV
us3	STRING[64]	Third user string variable USV

Examples

© 2025 Robox SpA

Example of use:

```
; definition of variable s_al of type STRU_AS_D
STRU_AS_D_EXT s_al

; I read the information about the alarm, if any, in the stack
as_get_d(1, s_al)

; I perform operations on the read data
; I check whether there is alarm 2000 at the position read with the
first string user parameter equal to USV
if ((s_al.alnum = 2000) and (s_al.us1 = "USV"))
    ...
endif
```

Structure STRU_AXIO_PDI

This keyword allows a variable of type STRU_AXIO_PDI to be defined.

This variable is used by the axio_robj() and axio_wobj() functions to read/write fieldbus objects.

It consists of the following information:

Field	Type	Description
rid	I32	Robox ID associated with the device: • 0, for AXIOLINE on RPx • >0, for remote AXIOLINE
slot	I32	Slot number of the module. 1 is the first module after RPx or BK EC
index	I32	Object index to read or write
subIndex	I32	Subindex of the object to read or write
datatype	I32	Type of data to be read or written (see codes table below). If 0, the data type is detected directly by the involved device. This variable is written by instruction with the data type used
datalen	I32	If used for AXIO_ROBJ: maximum size in bytes of the area used to store the response.

© 2025 Robox SpA

Field	Type	Description
		If used for AXIO_WOBJ: size in bytes of the data to be written when writing OCTECT_STRING or VISIBLE_STRING
phase	I32	Working phase/execution mode: • 1, passing execution initialization (**) • -1, blocking execution initialization (**) If used for AXIO_ROBJ: • 2-:-10, reserved RTE • 11, operation successfully executed If used for AXIO_WOBJ: • 2-:-6, reserved RTE • 7, operation executed successfully • 0, not initialized or error • other, illegal value **NOTE**: Values marked with (**) are the only ones that can be written to this variable by the user.
nric	I32	Number of characters received
res_1	I32	Reserved
res_2	I32	Reserved
res_3	I32	Reserved
res_4[256]	U8	Reserved

DataType codes table

Code	DataType
0	AUTO-SELECTION (default)
2	INTEGER8
3	INTEGER16
4	INTEGER32
5	UNSIGNED8

© 2025 Robox SpA

Code	DataType
6	UNSIGNED16
7	UNSIGNED32
9	VISIBLE_STRING
10	OCTET_STRING
21	INTEGER64
27	UNSIGNED64
-2	ARRAY INTEGER8
-3	ARRAY INTEGER16
-4	ARRAY INTEGER32
-5	ARRAY UNSIGNED8
-6	ARRAY UNSIGNED16
-7	ARRAY UNSIGNED32
-10	OCTET_STRING (used for special structures (e.g., parameter table CNT2/INC2)

Examples

Example declaration:

```
; definition of the variable s of type STRU_COE_OBJ
STRU_AXIO_PDI s
```

Structure STRU_BCC_TCP_CLIENT

Validity: **RC3E v33.9.1-17** (or later)

This keyword allows a variable of type STRU_BCC_TCP_CLIENT to be defined.

This variable is used by the R3 function bcc_get_tcp_clients_info 200() to obtain information regarding the list of registered clients and to know if the BCC message flow from the specific client is still active or not.

It consists of the following information:

Field	Type	Description
ip_addr	U32	IP address of the client
port	U16	Client port
status	U32	Logical status of the connection.

© 2025 Robox SpA

Field	Type	Description
		<table><tr><th>Bit</th><th>Mask</th><th>Meaning</th></tr><tr><td>0</td><td>0x00000001</td><td>Client is connected</td></tr><tr><td>1</td><td>0x00000002</td><td>Communication flow active</td></tr><tr><td>2</td><td>0x00000004</td><td>Communication flow missing (latched)</td></tr></table>
description	STRING [66]	Client description
last_access	REAL	Last valid time to receive a BCC message before setting "missing flow" [ms]
timeout	U32	Scheduled timeout [ms]
flags	U32	Reserved

Examples

Example declaration:

```
; definition of myvariable variable of type STRU_BCC_TCP_CLIENT
STRU_BCC_TCP_CLIENT myvariable
```

Esempio di utilizzo:

```
...
; I read ip address of client i and save it in an array
cl_ip[i] = myvariable[i].ip_addr
...
```

Structure STRU_CAM

This keyword allows a variable of type STRU_CAM (STRUCT_P 889) to be defined.

It is used by the mv_table 455() instruction, which allows you to run a cam for points.

It consists of the following information:

Field	Type	Description
x	REAL	x coordinate
y	REAL	y coordinate

Examples

Example declaration:

```
; definition of profile23 variable of type STRU_CAM
STRU_CAM profile23
```

Example of use:

```
; Use of the structure for a cam movement
; definition of variable profile23 of type STRU_CAM with seven pairs
of data
STRU_CAM profile23[7]
; data entry into the structure
if (not (r(100)))
    profile23[0].x = 0
    profile23[1].x = 90
    profile23[2].x = 135
    profile23[3].x = 180
    profile23[4].x = 270
    profile23[5].x = 330
    profile23[6].x = 360
    profile23[0].y = 10
    profile23[1].y = 90
    profile23[2].y = 250
    profile23[3].y = 300
    profile23[4].y = 270
    profile23[5].y = 100
    profile23[6].y = 10
    r(100) = 1
endif
...
; execution of cam movement
ip(2) = mv_table(ip(1) , 7 , profile23)
```

Structure STRU_CO_OBJ

This keyword allows a variable of type STRU_CO_OBJ to be defined.

This variable is used by the co_robj() and co_wobj() functions to read/write fieldbus objects.

It consists of the following information:

Field	Type	Description
rid	I32	Robox ID associated with the device
objCode	I32	Object number
subIndex	I32	Subindex number
type	I32	Data type according to the encoding used for co_send_sdo()/co_asw_sdo():

© 2025 Robox SpA

Field	Type	Description
		• 0, No object • 1, Unsigned 8 bits • 2, Unsigned 16 bits • 3, Unsigned 24 bits • 4, Unsigned 32 bits • 5, Real 32 bit • -1, Signed 8 bits • -2, Signed 16 bits • -3, Signed 24 bits • -4, Signed 32 bits
data	REAL	Data received/sent
phase	I32	Working phase/execution mode: • 1, passing execution initialization (**) • -1, blocking execution initialization (**) • 2, reserved RTE • 3, reserved RTE • 4, operation executed successfully • 0, not initialized or error • other, illegal value **NOTE**: Values marked with (**) are the only ones that can be written to this variable by the user.
res_1	I32	Reserved

Examples

Example declaration:

```
; definition of variable s of type STRU_CO_OBJ
STRU_CO_OBJ s
```

© 2025 Robox SpA

Structure STRU_COE_OBJ

Validity: **RC3E v33.5.18-2** (or later)

This keyword allows you to define a variable of type STRU_COE_OBJ.

This variable is used by the coe_robj 255() and coe_wobj 259() functions to read/write fieldbus objects.

It consists of the following information:

Field	Type	Description
rid	I32	Robox ID associated with the device
objCode	I32	Object number
subIndex	I32	Subindex number
datatype	I32	DataType (see codetable 807 below)
datalen	I32	If used for COE_ROBJ: maximum size in bytes of the area to store the response. If used for COE_WOBJ: size in bytes of the data to be written
phase	I32	Working phase/execution mode: • 1, passing execution initialization (**) • -1, blocking execution initialization (**) • 2, reserved RTE • 3, reserved RTE • 4, operation executed successfully • 0, not initialized or error • other, illegal value **NOTE**: Values marked with (**) are the only ones that can be written to this variable by the user.
res_1	I32	Reserved (number of characters received)
res_2	I32	Reserved
res_3	I32	Reserved

DataType codes table

© 2025 Robox SpA

Code	DataType
1	BOOLEAN
2	INTEGER8
3	INTEGER16
4	INTEGER32
5	UNSIGNED8
6	UNSIGNED16
7	UNSIGNED32
8	REAL32
9	VISIBLE_STRING
10	OCTET_STRING
11	UNICODE_STRING
12	TIME_OF_DAY
13	TIME_DIFFERENCE
15	DOMAIN
16	INTEGER24
17	REAL64
18	INTEGER40
19	INTEGER48
20	INTEGER56
21	INTEGER64
22	UNSIGNED24
24	UNSIGNED40
25	UNSIGNED48
26	UNSIGNED56
27	UNSIGNED64
30	BYTE
45	BITARR8
46	BITARR16
47	BITARR32

Examples

© 2025 Robox SpA

Example declaration:

```
; definition of variable s of type STRU_COE_OBJ
STRU_COE_OBJ s
```

Structure STRU_COERRMSG

This keyword allows a variable of type STRU_COERRMSG (STRUCT_P 889) to be defined.

This variable is used by the co_rerr 231() instruction for handling CanOpen messages.

It consists of the following information:

Field	Type	Description
errCode	U16	Emergency error code ()
errReg	U8	Error register
mse[5]	U8	Manufacturer specific error field byte 1, 2, 3, 4, 5

CiA Draft Standard Specifications

The following are the CiA Draft Standard specifications:

Emergency object data

The emergency telegram consists of 8 bytes with the data as shown below:

Byte	Content
0	Emergency Error Code (see table below)
1	
2	Error register (Object 1001H)
3	Manufacturer specific Error Field
4	
5	
6	
7	

Emergency Error Codes

Error Code (hex)	Meaning
00xx	Error Reset or No Error
10xx	Generic Error
20xx	Current

© 2025 Robox SpA

Error Code (hex)	Meaning
21xx	Current, device input side
22xx	Current, inside the device
23xx	Current, device output side
30xx	Voltage
31xx	Mains Voltage
32xx	Voltage inside the device
33xx	Output Voltage
40xx	Temperature
41xx	Ambient Temperature
42xx	Device Temperature
50xx	Device Hardware
60xx	Device Software
61xx	Internal Sw
62xx	User Sw
63xx	Data Set
70xx	Additional Modules
80xx	Monitoring
81xx	Communication
8110	CAN Overrun (Objects lost)
8120	CAN in Error Passive Mode
8130	Life Guard Error or Heartbeat Error
8140	Recovered from bus off
8150	Transmit COB-ID collision
82xx	Protocol Error
8210	PDO not processed due to length error
8220	PDO length exceeded
90xx	External Error
F0xx	Additional Functions
FFxx	Device specific

© 2025 Robox SpA

Structure of the Error Register

Bit	M/O	Meaning
0	M	Generic Error
1	O	Current
2	O	Voltage
3	O	Temperature
4	O	Communication Error (overrun, error state)
5	O	Device profile specific
6	O	Reserved (always 0)
7	O	Manufacturer specific

Examples

Example definition:

```
; definition of the variable analysisCan of type STRU_COERRMSG.
STRU_COERRMSG analysisCan
```

Example of use:

```
; use of the data contained in the structure.
; definition of the variable analysisCan of type STRU_COERRMSG.
STRU_COERRMSG analysisCan
; define a structure "MIA_STRUCT" of type STRUCT_P 889
STRUCT_P MY_STRUCT
U16 code_emg
U8 reg_error
U8 error_spec[5]
END_STRUCT
; I define the variable can_err of type MIA_STRUCT
MIA_STRUCT can_err
; I define the variable return and the variable can_wks
I32 returns
I32 can_wks
; I read the information about the possible alarm of the station can
can_wks = 1
returns = co_rerr 231(can_wks, analysisCan)
; I transfer the contents of the fields of the structure analysisCan
; into the can_err structure
move_byte 704(analysisCan, can_err, sizeof(can_err))
; I perform operations on the read data
if (can_err.code_emg = 0x1000)
 sr(1) = "GENERIC ERROR"
endif
```

© 2025 Robox SpA

Structure STRU_CRIMPER

This keyword allows a variable of type STRU_CRIMPER (STRUCT_P 889) to be defined.

It is used by the mv_crimper 427() motion function.

It consists of the following information:

Field	Type	Description
mastercontactlength	REAL	Length of the contact [0--*masterCycle*] in units of the master
mastercycle	REAL	Master cycle length
qMaster	REAL	Current position of the master. To be always refreshed before each call to the function
qSlave	REAL	Current position of the slave. To be always refreshed before each call to the function
reservedCrimper1	REAL	Reserved
reservedCrimper2	REAL	Reserved
status	I32	Motion status. • M_INITIALIZING, on the first call of the function the user must set the status to this one • M_HOMING, if the recovery mission is in progress • M_SYNCHRONIZED, if the slave is docked to the master Refer to the returns codes 61 for the relevant codes.
slavedirection	REAL	Slave axis direction
slavephase	REAL	Phase of the slave in unit of the master. If from 10 to 20, the difference (+10) is added to *mastercycle* (one cycle only). With *slavephase* = 0, there is a binding such that whenever a new master cycle starts, the crimper axis is at the beginning of its docking zone
slavecycle	REAL	Slave cycle length in slave's units

© 2025 Robox SpA

Field	Type	Description
kslavemaster	REAL	K factor of master/slave unit of measure. If master unit is in mm and slave unit is in mm, then K=1. If master unit is in mm and slave unit is in degrees, for example, if every 500 mm of master advance the blade must make 360 degrees then *kslavemaster* = 2 * K_PI 17 * blade radius / 360

Examples

Example declaration:

```
; definition of variable parGan1 of type STRU_CRIMPER
STRU_CRIMPER parGan1
```

Example of use:

```
; use of data contained in structure
; definition of variable parGan1 of type STRU_CRIMPER
STRU_CRIMPER parGan1
; definition of accessory variables
I32 result
REAL returns
; insertion of data into the structure
if (not(r(100)))
    parGan1.mastercontactlength = 500
    parGan1.mastercycle = 2000
    parGan1.status = 1 ; 1 is the code for M_INITIALIZING
    parGan1.slavephase = 25
    parGan1.slavecycle = 800
    parGan1.kslavemaster = 2
    r(100) = 1
endif
; before calling the mv_crimper() function.
; I update the shares of the master and the slave
parGan1.qMaster = cp(1)
parGan1.qSlave = cp(2)
; I execute the mv_crimper() function.
returns = mv_crimper 427(result, parGan1)
; I evaluate the variable result
if(result = 8) ; 8 is the code for M_SYNCHRONIZED
    sr(1) = "I am synchronized"
endif
```

© 2025 Robox SpA

Structure STRU_ECAT_PROBE

Validity: **RC3E v33.7-1-beta0** (or later).

This keyword defines a variable of type STRU_ECAT_PROBE.

It consists of the following information:

Field	Type	Description
phase	U32	Internal working phase. Must be set to 0 at the start
err	I32	Internal error code: • -14 (EFAULT) null pointer or illegal phase • -5 (EIO) cable not connected • -2 (ENOENT) no device found • -19 (ENODEV) selected device does not exist
n_pos	U32	Ordinal position in the ethercat line (1-:-1024)
how_many	U32	Number of devices present on the ethercat line
al_status	U16	Returns: • 0x01 Init • 0x02 Pre-Operational • 0x03 Bootstrap • 0x04 Safe-Operational • 0x08 Operational • 0x11 Init + err • 0x12 Pre-Operational + err • 0x13 Bootstrap + err • 0x14 Safe-Operational + err • 0x18 Operational + err
al_status_code	U16	AL status code of the device
config_addr	U16	Configured device address
alias_addr	U16	Configured device address
vendor_id	U32	Vendor ID
product_code	U32	Product code of the device
revision_number	U32	Revision number of the device in EEP

© 2025 Robox SpA

Field	Type	Description
serial_number	U32	Serial number of the device in EEP

Examples

Example declaration:

```
; definition of handle variable of type STRU_ECAT_PROBE
STRU_ECAT_PROBE sP
```

Structure STRU_FILEHND

This keyword allows a variable of type STRU_FILEHND (STRUCT_P 889) to be defined.

It is used by the flash management instructions f_open 302(), f_close 294(), f_read 305(), f_gets 300(), ...

It consists of the following information:

Field	Type	Description
h	I32	handle

Examples

Example declaration:

```
; definition of handle variable of type STRU_FILEHND
STRU_FILEHND handle
```

Example of use:

```
; use of data contained in the structure with the f_open() function.
; definition of handle variable of type STRU_FILEHND
STRU_FILEHND handle
; definition of auxiliary veriables
STRING file_name ;  is the name of the file to open
STRING mode_open ;  is the mode of opening
I32 car_max = 0 ; must be zero
I32 returns ; result of the operation
; I assign a value to the variables
filename = sr(1)
mode_open = sr(2)
; I execute the file open function
returns = f_open 302(file_name, mode_open, car_max, handle)
; I perform operations on the read data
if (returns)
    sr(3) = "I opened the file " # filename
else
    sr(3) = "The file " # filename # " does not exist!"
endif
```

© 2025 Robox SpA

Structure STRU_FILEINFO

This keyword allows a variable of type STRU_FILEINFO (STRUCT_P 889) to be defined.

It is used by the f_dir 297() function to read information about files contained in flash memories.

It consists of the following information:

Field	Type	Description
reserved[16]	STRING	Reserved
len	U32	File length
name[128]	STRING	File name

Examples

Example definition:

```
; definition of archive variable of type STRU_FILEINFO
STRU_FILEINFO archive
```

Example of use:

```
; use of data contained in the structure.
; definition of archive variable of type STRU_FILEINFO
STRU_FILEINFO archive
; definition of auxiliary veriables
STRING bank_f ;  is the name of the bank of the flash
I32 num_file ;  is the number of files to be read
I32 returns ; result of the operation
; I assign a value to the variables
bank_f = "/FA/*.*"
; I use the value saved in register 220
num_file = r(220)
; I execute the file open function
returns = f_dir 297(bank_f, num_file, archive)
; I perform operations on the read data
if (returns)
    sr(3) = "There are " # str_i 602(returns, 1, 10) # " files in the
flash"
else
    sr(3) = "The flash is empty!"
endif
```

Structure STRU_FOLLOW

This keyword allows a variable of type STRU_FOLLOW (STRUCT_P 889) to be defined.

It is used by the mv_follow 428() motion function.

It consists of the following information:

Field	Type	Description
masterDirection	REAL	• 1, if the master axis evolves toward positive quotas • -1, if the master axis evolves toward negative quotas
qSyncMaster	REAL	Master axis synchronism quota
masterCycle	REAL	Increment step of the *qSyncMaster* dimension to handle repeated cyclic tracking (unsigned)
qMaster	REAL	Master position. Must be updated before each function call
velMaster	REAL	Master speed. Must be updated before each function call
si	REAL	Sampling interval si
slaveDirection	REAL	• 1, if the slave axis evolves toward positive quotas • -1, if the slave axis evolves toward negative quotas
qSyncSlave	REAL	Slave axis synchronism start quota
qSyncOut	REAL	Slave axis synchronism stop quota
qParkSlave	REAL	Parking quota of the slave axis
slaveCycle	REAL	*qSyncSlave* quota increment step to handle repeated cyclic tracking (unsigned)
velMaxSlave	REAL	This is the upper speed limit allowed for slave axis movement. In case the correction formula requires going above the maximum speed, the value returned by the function would be equal to *velMaxSlave*
accMaxSlave	REAL	It is the maximum acceleration value accepted for the slave axis
kSlaveMaster	REAL	K of master/slave adaption. e.g. if a knife 250 mm long, *kslavemaster* = K_DUEPI 17 * 0.25 / 360

© 2025 Robox SpA

Field	Type	Description
qSlave	REAL	Slave position. Must be updated before each function call
velSlave	REAL	Slave speed. Must be updated before each function call
status	I32	• M_POSITIONING, the slave axis is moving to the parking position. On the first call of the function, the user must set status to M_POSITIONING • MV_WAITING, the slave axis is waiting to start • M_WAITING_NEXT_CYCLE, there are no conditions to leave; I am waiting for the next appointment • M_REACHED, the slave axis has hooked up Refer to the returns codes 61 for the relevant codes.

Examples

Example declaration:

```
; definition of variable saw1 of type STRU_FOLLOW
STRU_FOLLOW saw1
```

Example of use:

```
; use of the data contained in the structure.
; definition of variable saw1 of type STRU_FOLLOW
STRU_FOLLOW saw1
; definition of accessory variables
I32 result
REAL returns
; insertion of data into the structure
if(not(r(100)))
    saw1.masterDirection = 1
    saw1.qSyncMaster = 400
    sawa1.masterCycle = 50
    sawa1.si = si
    sawa1.slaveDirection = 1
    sega1.qSyncSlave = 80
    sawa1.qSyncOut = 300
    sawa1.qParkSlave = 250
    sawa1.slaveCycle = 30
    sawa1.velMaxSlave = 60
    sawa1.accMaxSlave = 10
```

© 2025 Robox SpA

```
    sawa1.kSlaveMaster = 2
    sega1.status = 2 ; 2 is the code for M_POSITIONING
    r(100) = 1
endif
; before calling the mv_follow() function.
; I update the dimensions and speed of the master and slave
saw1.qMaster = cp(1)
sawa1.velMaster = cv(1)
sawa1.qSlave = cp(2)
sawa1.velSlave  =cv(2)
; I run the mv_follow() function.
returns = mv_follow(result, saw1)
; I evaluate the variable result
if(result = 6) ; 6 is the code for M_REACHED
    sr(1) = "Axis hooked"
endif
```

Structure STRU_FOLLOW2

Validity: **RC3E v33.5.8-7** (or later).

This keyword allows a variable of type STRU_FOLLOW2 (STRUCT_P) to be defined.

It is used by the mv_follow2() motion function.

It consists of the following information:

Field	Type	Description
command	U32	Command and configuration word:

Bit	Hexadecimal value	Description
Configurable runtime commands:		
0	0x00000001	If set to 1 there is a slave stop at *qParkSlave* during the next cycle (the next cycle is not executed) (active on level)
1	0x00000002	If set to 1 there is initial positioning

Field	Type	Description

Bit	Hexadecimal value	Description
		at *qParkSlave* (auto-reset by RTE)
2	0x00000004	If set to 1 you have the initial positioning is done with the master cycle (auto-reset by RTE)
3	0x00000008	Forcing the end of the synchronism segment (auto-reset by RTE)
4	0x00000010	If linear type slave, forces stop at end of forward cycle as long as such flag is present (active on level)
5	0x00000020	Indicates that the supplied master sync position is not yet the real one (only assumed). Useful if used with OB ELEMENT LOCATION. In that case, the

© 2025 Robox SpA

Field	Type	Description
		<table><tr><th>Bit</th><th>Hexadecimal value</th><th>Description</th></tr><tr><td></td><td></td><td>next docking is not performed and the slave remains still (active on level)</td></tr><tr><td colspan="3">Configuration flag:</td></tr><tr><td>16</td><td>0x00010000</td><td>• 0, fixed-cycle master(*masterCycle* = cycle length, *qSyncMaster* = sync quota relative to the cycle) • 1, variable-cycle master(*masterCycle* = minimum cycle length, *qSyncMaster* = absolute sync quota updated at each cycle, e.g., on a photocell reading)</td></tr><tr><td>17</td><td>0x00020000</td><td>If I set to 1 there is a disabling of the test on *maxSpe*</td></tr><tr><td>18</td><td>0x00040000</td><td>If I set to 1 you have</td></tr></table>

© 2025 Robox SpA

Field	Type	Description
		<table><tr><th>Bit</th><th>Hexadecimal value</th><th>Description</th></tr><tr><td></td><td></td><td>the disabling of the test on *maxAcc*</td></tr><tr><td>19</td><td>0x00080000</td><td>• 0, *qSyncMaster* aligned to *qSyncSlave* • 1, *qSyncMaster* aligned to the midpoint between *qSyncSlave* and *qOutSlave*</td></tr><tr><td>20</td><td>0x00100000</td><td>Valid only for cyclic slave*(slaveCycle>0)*. • 1, enables slave stop at *qParkSlave* at each cycle for the duration *spaceWaitInPark* In case of linear slave*(slaveCycle=0)* the stop at park quota is implied</td></tr><tr><td>21</td><td>0x00200000</td><td>Valid only for cyclic slave*(slaveCycle>0)*. • 0, slave can also</td></tr></table>

© 2025 Robox SpA

<table>
<tr><th>Field</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><th>Bit</th><th>Hexadecimal value</th><th>Description</th></tr>
<tr><td></td><td></td><td>move backward (during positioning)
• 1, slave can move forward only</td></tr>
<tr><td>22</td><td>0x00400000</td><td>Valid only for cyclic slave(slave Cycle>0), variable cycle master (BIT 16=1) and stop at qParkSlave disabled (BIT 20=0)
• 1, Enables the recover of the synchronism during the slowdown ramp</td></tr>
</table>
</td></tr>
<tr><td>errorCode</td><td>U32</td><td>Error codes (see error code table 825 below)</td></tr>
<tr><td colspan="3">MASTER DATA:</td></tr>
<tr><td>masterCycle</td><td>REAL</td><td>Master cycle step if fixed-cycle master (B16= 0).
Minimum master cycle step if variable-cycle master (B16=1).
NOTE: If variable-cycle master can also be left at 0, in which case, however, no checks are made on the recovery speed.</td></tr>
<tr><td>qSyncMaster</td><td>REAL</td><td>Master quota used as a synchronism reference.
If fixed-cycle master (B16=0) it is assumed as a relative quota for</td></tr>
</table>

Field	Type	Description
		the cycle and should be set once. If variable-cycle master (B16=1) it is an absolute quota that should be reset at each cycle, e.g, on the photocell
qMaster	REAL	Master position. It must always be updated before each function call. If axis it corresponds to the IP
velMaster	REAL	Master speed. It must always be updated before each function call. If axis it corresponds to the IV
accMaster	REAL	Master acceleration. It must always be updated before each function call. If axis it corresponds to the AI. **NOTE**: This parameter is only used to correctly calculate *accSlave*.
SLAVE DATA:		
slaveCycle	REAL	Step increment of *qSyncSlave* quota to handle repeated-cyclic following. (if linear slave should be left at 0)
qSyncSlave	REAL	Synchronism start quota of slave axis
qOutSlave	REAL	Synchronism final quota of slave axis
qParkSlave	REAL	Parking quota of the slave axis
spaceStopSlave	REAL	Stop space of the slave from *qOutSlave*. Valid only if linear slave (slaveCycle = 0)
spaceWaitInPark	REAL	Master advance space during which the slave axis remains stationary at parking quota *qParkSlave* at each cycle before starting the next tracking. This parameter is only valid if the slave is linear (slaveCycle = 0) or if the slave is cyclic and the stop at quota *qParkSlave* is enabled at each cycle (B20=1)
velPositSlave	REAL	Speed initial positioning on *qParkSlave*

© 2025 Robox SpA

Field	Type	Description
accPositSlave	REAL	Acceleration initial positioning on *qParkSlave*
jerkPositSlave	REAL	Jerk initial positioning on *qParkSlave*
kSlaveMaster	REAL	Master speed to slave speed conversion constant used during the synchronized segment (*velSlave* = *velMaster* * *kSlaveMaster*). **NOTE**: It is used to equalize units if they are different, or to differentiate speeds during the synchronized segment.
qSlave	REAL	Current slave quota updated runtime by the function (the first time the function is used it should be initialized to the actual value at which the slave axis is)
vSlave	REAL	Current slave speed updated runtime by the function
aSlave	REAL	Current slave acceleration updated runtime by the function. **NOTE**: Its value is correct only if the *accMaster* parameter is set correctly.
maxSpe	REAL	It is the maximum speed value for the movement of the slave. Used if relative test is enabled (B17=0)
maxAcc	REAL	Is the maximum acceleration value for the movement of the slave. Used if relative test enabled (B18=0)
INTERNAL STORAGE:		
resDouble[14]	REAL	Reserved
resInt[2]	REAL	Reserved

Error codes table

Value	Meaning
< 0	Internal error (contact Robox)

© 2025 Robox SpA

Value	Meaning
0	No error
1	Invalid *maxAcc*
2	Insufficient start or stop slave space
3	Invalid *kSlaveMaster*
4	Insufficient start master space + brake space
5	Insufficient recovery master space
6	Master cycle not set with fixed cycle master
7	Unable to perform positioning (kinematic positioning parameters invalid)
8	Exceeding maximum speed in tracking phase
9	Exceeding maximum acceleration in tracking phase
10	Exceeding maximum speed in synchronism phase
11	Exceeding maximum acceleration in synchronism phase
12	Exceeding maximum speed in stopping phase
13	Exceeding maximum acceleration in stopping phase
14	Exceeding maximum velocity in homing phase
15	Exceeding maximum acceleration in homing phase

Examples

Example of use:

```
; definition of variable saw1 of type STRU_FOLLOW2
STRU_FOLLOW2 saw1
result = mv_follow2 432 (saw1)
set_kinematics 569 (ax1, saw1.qSlave, saw1.velSlave, saw1.accSlave)
```

Structure STRU_FOLLOW2_INFO

Validity: **RC3E v33.5.8-7** (or later).

This keyword allows a variable of type STRU_FOLLOW2_INFO (STRUCT_P 889) to be defined.

It is used by the mv_follow2 432() motion function.

It consists of the following read-only information:

Field	Type	Description
qmastertostart	REAL	Missing space for the master to reach the tracking start quota
qmastertosync	REAL	Missing space for the master to reach the synchronism start quota
qmastertoout	REAL	Missing space for the master to reach the synchronism final quota

Examples

Example of use:

```
; definition of variable infoShear of type STRU_FOLLOW2_INFO
STRU_FOLLOW2_INFO infoShear
timetoStart = infoShear.qmastertostart / IV(master)
```

Structure STRU_FOLLOW2_S

Validity: **RC3E v33.5.8-7** (or later).

This keyword allows a variable of type STRU_FOLLOW2_S (STRUCT_P 889) to be defined.

It is used by the mva_follow2 432() motion function.

It consists of the following information:

Field	Type	Description
nAxS	U32	Slave axis number
command	U32	Command and configuration word: Bit | Hexadecimal value | Description Configurable runtime commands: 0 | 0x00000001 | If set to 1 there is a slave stop at *qParkSlave* during the

Field	Type	Description
		<table><tr><th>Bit</th><th>Hexadecimal value</th><th>Description</th></tr><tr><td></td><td></td><td>next cycle (the next cycle is not executed) (active on level)</td></tr><tr><td>1</td><td>0x00000002</td><td>If set to 1 there is initial positioning at *qParkSlave* (auto-reset by RTE)</td></tr><tr><td>2</td><td>0x00000004</td><td>If set to 1 you have the initial positioning is done with the master cycle (auto-reset by RTE)</td></tr><tr><td>3</td><td>0x00000008</td><td>Forcing the end of the synchronism segment (auto-reset by RTE)</td></tr><tr><td>4</td><td>0x00000010</td><td>If linear type slave, forces stop at end of forward cycle as long as such flag is present (active on level)</td></tr><tr><td>5</td><td>0x00000020</td><td>Indicates that the supplied master sync position is</td></tr></table>

© 2025 Robox SpA

Field	Type	Description
		<table><tr><th>Bit</th><th>Hexadecimal value</th><th>Description</th></tr><tr><td></td><td></td><td>not yet the real one (only assumed). Useful if used with OB ELEMENT LOCATION. In that case, the next docking is not performed and the slave remains still (active on level)</td></tr><tr><td colspan="3">Configuration flag:</td></tr><tr><td>16</td><td>0x00010000</td><td>• 0, fixed-cycle master(*masterCycle* = cycle length, *qSyncMaster* = sync quota relative to the cycle) • 1, variable-cycle master(*masterCycle* = minimum cycle length, *qSyncMaster* = absolute sync quota updated at each cycle, e.g., on a</td></tr></table>

© 2025 Robox SpA

Field	Type	Description
		<table><tr><th>Bit</th><th>Hexadecimal value</th><th>Description</th></tr><tr><td></td><td></td><td>photocell reading)</td></tr><tr><td>17</td><td>0x00020000</td><td>If I set to 1 there is a disabling of the test on *maxSpe*</td></tr><tr><td>18</td><td>0x00040000</td><td>If I set to 1 you have the disabling of the test on *maxAcc*</td></tr><tr><td>19</td><td>0x00080000</td><td>• 0, *qSyncMaster* aligned to *qSyncSlave* • 1, *qSyncMaster* aligned to the midpoint between *qSyncSlave* and *qOutSlave*</td></tr><tr><td>20</td><td>0x00100000</td><td>Valid only for cyclic slave(*slaveCycle>0*). • 1, enables slave stop at *qParkSlave* at each cycle for the duration *spaceWaitInPark* In case of linear slave(*slaveCycle=0*)</td></tr></table>

© 2025 Robox SpA

<table>
<tr><th>Field</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><th>Bit</th><th>Hexadecimal value</th><th>Description</th></tr>
<tr><td></td><td></td><td>the stop at park quota is implied</td></tr>
<tr><td>21</td><td>0x00200000</td><td>Valid only for cyclic slave(slave Cycle>0).
• 0, slave can also move backward (during positioning)
• 1, slave can move forward only</td></tr>
<tr><td>22</td><td>0x00400000</td><td>Valid only for cyclic slave(slave Cycle>0), variable cycle master (BIT 16=1) and stop at qParkSlave disabled (BIT 20=0)
• 1, Enables the recover of the synchronism during the slowdown ramp</td></tr>
</table>
</td></tr>
<tr><td>errorCode</td><td>U32</td><td>Error codes (see error code table 833 below)</td></tr>
<tr><td colspan="3">MASTER DATA:</td></tr>
<tr><td>masterCycle</td><td>REAL</td><td>Master cycle step if fixed-cycle master (B16= 0).
Minimum master cycle step if variable-cycle master (B16=1).</td></tr>
</table>

© 2025 Robox SpA

Field	Type	Description
		NOTE: If variable-cycle master can also be left at 0, in which case, however, no checks are made on the recovery speed.
qSyncMaster	REAL	Master quota used as a synchronism reference. If fixed-cycle master (B16=0) it is assumed as a relative quota for the cycle and should be set once. If variable-cycle master (B16=1) it is an absolute quota that should be reset at each cycle, e.g, on the photocell
qMaster	REAL	Master position. It must always be updated before each function call. If axis it corresponds to the IP
velMaster	REAL	Master speed. It must always be updated before each function call. If axis it corresponds to the IV
accMaster	REAL	Master acceleration. It must always be updated before each function call. If axis it corresponds to the AI. **NOTE**: This parameter is only used to correctly calculate *accSlave*.
SLAVE DATA:		
slaveCycle	REAL	Step increment of *qSyncSlave* quota to handle repeated-cyclic following. (if linear slave should be left at 0)
qSyncSlave	REAL	Synchronism start quota of slave axis
qOutSlave	REAL	Synchronism final quota of slave axis
qParkSlave	REAL	Parking quota of the slave axis
spaceStopSlave	REAL	Stop space of the slave from *qOutSlave*. Valid only if linear slave (slaveCycle = 0)
spaceWaitInPark	REAL	Master advance space during which the slave axis remains stationary at parking quota *qParkSlave* at each cycle before starting the next tracking.

© 2025 Robox SpA

Field	Type	Description
		This parameter is only valid if the slave is linear (slaveCycle = 0) or if the slave is cyclic and the stop at quota *qParkSlave* is enabled at each cycle (B20=1)
velPositSlave	REAL	Speed initial positioning on *qParkSlave*
accPositSlave	REAL	Acceleration initial positioning on *qParkSlave*
jerkPositSlave	REAL	Jerk initial positioning on *qParkSlave*
kSlaveMaster	REAL	Master speed to slave speed conversion constant used during the synchronized segment (*velSlave* = *velMaster* * *kSlaveMaster*). **NOTE**: It is used to equalize units if they are different, or to differentiate speeds during the synchronized segment.

Error codes table

Value	Meaning
< 0	Internal error (contact Robox)
0	No error
1	Invalid *maxAcc*
2	Insufficient start or stop slave space
3	Invalid *kSlaveMaster*
4	Insufficient start master space + braking master space
5	Insufficient recovery master space
6	Master cycle not set with fixed cycle master
7	Unable to perform positioning (invalid positioning kinematic parameters)
8	Exceeding maximum speed in tracking phase
9	Exceeding maximum acceleration in tracking phase

© 2025 Robox SpA

Value	Meaning
10	Exceeding maximum speed in synchronism phase
11	Exceeding maximum acceleration in synchronism phase
12	Exceeding maximum speed in stop phase
13	Exceeding maximum acceleration in stop phase
14	Exceeding maximum speed in homing phase
15	Exceeding maximum acceleration in homing phase

Examples

Example of use:

```
; definition of variable mf2 of type STRU_FOLLOW2_S
STRU_FOLLOW2_S mf2
result = mva_follow2(mf2)
```

Structure STRU_FOLLOW2_SM

Validity: **RC3E v33.5.8-7** (or later).

This keyword allows a variable of type STRU_FOLLOW2_SM (STRUCT_P) to be defined.

It is used by the movement instruction mva_follow2_m().

It consists of the following information:

Field	Type	Description
nAxM	U32	Master axis number
nAxS	U32	Slave axis number
command	U32	Command and configuration word:

Bit	Hexadecimal value	Description
Configurable runtime commands:		
0	0x00000001	If set to 1 there is a slave stop at *qParkSlave* during the

Field	Type	Description

Bit	Hexadecimal value	Description
		next cycle (the next cycle is not executed) (active on level)
1	0x00000002	If set to 1 there is initial positioning at *qParkSlave* (auto-reset by RTE)
2	0x00000004	If set to 1 you have the initial positioning is done with the master cycle (auto-reset by RTE)
3	0x00000008	Forcing the end of the synchronism segment (auto-reset by RTE)
4	0x00000010	If linear type slave, forces stop at end of forward cycle as long as such flag is present (active on level)
5	0x00000020	Indicates that the supplied master sync position is

© 2025 Robox SpA

Field	Type	Description
		<table><tr><th>Bit</th><th>Hexadecimal value</th><th>Description</th></tr><tr><td></td><td></td><td>not yet the real one (only assumed). Useful if used with OB ELEMENT LOCATION. In that case, the next docking is not performed and the slave remains still (active on level)</td></tr><tr><td colspan="3">Configuration flag:</td></tr><tr><td>16</td><td>0x00010000</td><td>• 0, fixed-cycle master(*masterCycle* = cycle length, *qSyncMaster* = sync quota relative to the cycle) • 1, variable-cycle master(*masterCycle* = minimum cycle length, *qSyncMaster* = absolute sync quota updated at each cycle, e.g., on a</td></tr></table>

© 2025 Robox SpA

Field	Type	Description

Bit	Hexadecimal value	Description
		photocell reading)
17	0x00020000	If I set to 1 there is a disabling of the test on *maxSpe*
18	0x00040000	If I set to 1 you have the disabling of the test on *maxAcc*
19	0x00080000	• 0, *qSyncMaster* aligned to *qSyncSlave* • 1, *qSyncMaster* aligned to the midpoint between *qSyncSlave* and *qOutSlave*
20	0x00100000	Valid only for cyclic slave*(slaveCycle>0)*. • 1, enables slave stop at *qParkSlave* at each cycle for the duration *spaceWaitInPark* In case of linear slave*(slaveCycle=0)*

<table>
<tr><th>Field</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><th>Bit</th><th>Hexadecimal value</th><th>Description</th></tr>
<tr><td></td><td></td><td>the stop at park quota is implied</td></tr>
<tr><td>21</td><td>0x00200000</td><td>Valid only for cyclic slave(slave Cycle>0).
• 0, slave can also move backward (during positioning)
• 1, slave can move forward only</td></tr>
<tr><td>22</td><td>0x00400000</td><td>Valid only for cyclic slave(slave Cycle>0), variable cycle master (BIT 16=1) and stop at qParkSlave disabled (BIT 20=0)
• 1, Enables the recover of the synchronism during the slowdown ramp</td></tr>
</table>
</td></tr>
<tr><td>errorCode</td><td>U32</td><td>Error codes (see error code table 840 below)</td></tr>
<tr><td colspan="3">MASTER DATA:</td></tr>
<tr><td>masterCycle</td><td>REAL</td><td>Master cycle step if fixed-cycle master (B16= 0).
Minimum master cycle step if variable-cycle master (B16=1).</td></tr>
</table>

Field	Type	Description
		NOTE: If variable-cycle master can also be left at 0, in which case, however, no checks are made on the recovery speed.
qSyncMaster	REAL	Master quota used as a synchronism reference. If fixed-cycle master (B16=0) it is assumed as a relative quota for the cycle and should be set once. If variable-cycle master (B16=1) it is an absolute quota that should be reset at each cycle, e.g, on the photocell
SLAVE DATA:		
slaveCycle	REAL	Step increment of *qSyncSlave* quota to handle repeated-cyclic following. (if linear slave should be left at 0)
qSyncSlave	REAL	Synchronism start quota of slave axis
qOutSlave	REAL	Synchronism final quota of slave axis
qParkSlave	REAL	Parking quota of the slave axis
spaceStopSlave	REAL	Stop space of the slave from *qOutSlave*. Valid only if linear slave (slaveCycle = 0)
spaceWaitInPark	REAL	Master advance space during which the slave axis remains stationary at parking quota *qParkSlave* at each cycle before starting the next tracking. This parameter is only valid if the slave is linear (slaveCycle = 0) or if the slave is cyclic and the stop at quota *qParkSlave* is enabled at each cycle (B20=1)
velPositSlave	REAL	Speed initial positioning on *qParkSlave*
accPositSlave	REAL	Acceleration initial positioning on *qParkSlave*
jerkPositSlave	REAL	Jerk initial positioning on *qParkSlave*
kSlaveMaster	REAL	Master speed to slave speed conversion constant used during

© 2025 Robox SpA

Field	Type	Description
		the synchronized segment (*velSlave* = *velMaster* * *kSlaveMaster*). **NOTE**: It is used to equalize units if they are different, or to differentiate speeds during the synchronized segment.

Error codes table

Value	Meaning
< 0	Internal error (contact Robox)
0	No error
1	Invalid *maxAcc*
2	Insufficient start or stop slave space
3	Invalid *kSlaveMaster*
4	Insufficient starting + braking master space
5	Insufficient recovery master space
6	Master cycle not set with fixed cycle master
7	Unable to perform positioning (invalid positioning kinematic parameters)
8	Exceeding maximum speed in tracking phase
9	Exceeding maximum acceleration in tracking phase
10	Exceeding maximum speed in synchronism phase
11	Exceeding maximum acceleration in synchronism phase
12	Exceeding maximum speed in stopping phase
13	Exceeding maximum acceleration in stopping phase
14	Exceeding maximum speed in homing phase
15	Exceeding maximum acceleration in homing phase

Examples

© 2025 Robox SpA

Example of use:

```
; definition of variable myShear of type STRU_FOLLOW2_SM
STRU_FOLLOW2_SM myShear
result = mva_follow2_m 432(myShear)
```

Structure STRU_GROR

This keyword allows a variable of type STRU_GROR (STRUCT_P 889) to be defined.

It is used by the group 694 and order 708 instructions, which are used to activate the rule executors (RULE) and define their order.

It consists of the following information:

Field	Type	Description
idx[1;32]	I32	Number of the Rule

Examples

Example of use:

```
; definition of the variable activeRule of type STRU_GROR
STRU_GROR activeRule
; rc(1) will be executed first
activeRule.idx[1] = 1
; rc(2) will be executed second
activeRule.idx[2] = 2
; rc from 3 to 32 will be disabled
activeRule.idx[3] = -1
; assign order
order 708(activeRule)
```

Structure STRU_HW_INFO

Validity: **RC3E v33.5.23-3** (or later).

This keyword allows a variable of type STRU_HW_INFO (STRUCT_P 889) to be defined.

It is used by the get_hw_info 326() function.

It consists of the following information:

Field	Type	Description
board_code_l	U32	Low board code
board_code_h	U32	High code of the board
iwow	I32	Number of the first input word and first output word associated with the board

© 2025 Robox SpA

Field	Type	Description
name	STRING[64]	Name of the board

Examples

Example of use:

```
; Using the get_hw_info statement with nSlot=0 we get:
name = "CPURBX"
iwow = 1
board_code_l = 0x13
board_code_h = 0x0
```

Structure STRU_INT

This keyword allows a variable of type STRU_INT (STRUCT_P 889) to be defined.

It is used by the interrupt handling instructions int_timer 701(), int_after 698(), int_inp 699().

It consists of the following information:

Field	Type	Description	Attributes
cntr	I32	• BIT 0: If set to 0 there is continuous execution of the interrupt. If set to 1 you have one-shot execution • BIT 1: If set to 0 it uses STRU_INT.count as the interrupt counter. If set to 1 it uses STRU_INT.count as a flag by setting it to 1 every time an interrupt occurs. It is up to the user to reset STRU_INT.count to zero	Read/write
count	I32	Interrupt counter	Read/write
intTime	REAL	Time [sec] relative to the occurrence of the interrupt. This time should be compared with the system parameter synch_hw_time [microsec], which corresponds to the time of the last system synch. **WARNING**: *intTime* is in [sec] while *synch_hw_time* is in [microsec]. To calculate	Read

Field	Type	Description	Attributes
		at what time distance the interrupt occurred from the last system synch use the formula: delta_sec = intTime - (synch_hw_time * 1e-6). This formula should be used within the rule called by the interrupt, in which case it will always be: 0 < delta_sec < si	
maskPhoto	I32	Mask of the axes whose photos are wanted on the occurrence of the interrupt	Read/write
nrule	I32	Number of rules to be activated (it must be contained in the $ rule_time file and/or in the $ rule_inp file)	Read/write
pos[1-32]	REAL	Photographed position of axes (enabled by *maskPhoto*) on interrupt	Read
sts	I32	• 1, the data structure is used by the instructions int_timer 701, int_inp 699, int_after 698 • 2, the data structure is used by the Rule_periodic 730 instruction	Read

Examples

Example definition:

```
; definition of box variable of type STRU_INT
STRU_INT box
```

Example of use:

```
; use of data contained in structure
; definition of box variable of type STRU_INT
STRU_INT box
; I write the data in the structure
box.cntr.0 = 0 ; continuous execution
box.cntr.1 = 0 ; totalizer is enabled to count
box.maskPhoto = 0x3 ; I enable the photos of the first two axes
box.nrule = 100 ; enable rule number 100
```

© 2025 Robox SpA

```
box.sts = 1 ; I indicate that the structure is used by the int_xxx
instruction
```

Structure STRU_LARGE_INT

This keyword allows a variable of type STRU_LARGE_INT to be defined.

This structure is to be used as a substitute for the i64 type not found in the R3 language.

It can be used in the get_i64 692() and set_i64 742() instructions.

It consists of the following information:

Field	Type	Description
high	I32	Upper part of the i64
low	U32	Lower part of the i64

Examples

Example of use:

```
U8 buffer[128]
I32 val_high
U32 val_low
STRU_LARGE_INT set_val
STRU_LARGE_INT my_data

; -------- loop begin --------
__main_loop__

    ; range check
    if (offset >= 0 and (offset + sizeof 576(my_data)) < 128)
        set_val.high = val_high
        set_val.low  = val_low
        set_i64 742(buffer, offset, set_val)
        get_i64 692(buffer, offset, my_data)
    endif

end_main_loop
```

Structure STRU_LARGE_UINT

This keyword allows a variable of type STRU_LARGE_UINT to be defined.

This structure is to be used as a substitute for the u64 type not found in the R3 language.

It can be used in the get_u64 693() and set_u64 747() instructions.

It consists of the following information:

© 2025 Robox SpA

Field	Type	Description
high	U32	Upper part of the u64
low	U32	Lower part of the u64

Examples

Example of use:

```
U8 buffer[128]
U32 val_high
U32 val_low
STRU_LARGE_UINT set_val
STRU_LARGE_UINT my_data

; -------- loop begin --------
__main_loop__

    ; range check
    if (offset >= 0 and (offset + sizeof(my_data)) < 128)
        set_val.high = val_high
        set_val.low  = val_low
        set_u64(buffer, offset, set_val)
        get_u64(buffer, offset, my_data)
    endif

end_main_loop
```

Structure STRU_MASTERF

Validity: **RC3E v33.5.8-7** (or later).

This keyword allows a variable of type STRU_MASTERF (STRUCT_P) to be defined.

It is used by the master_filter() function.

It consists of the following information:

Field	Type	Description
velFilterTime	REAL	Filtering speed time constant limited to be >= SI
errRecoveryTime	REAL	Error recovery time constant limited to be >= SI
filteredVel	REAL	Filtered speed (function storage, to be typically initialized to 0)
posErr	REAL	Position error (function storage, to be initialized typically to 0)

© 2025 Robox SpA

Field	Type	Description
masterPos	REAL	Real position of the master (to be set only if NAX = 0)
mastervel	REAL	Master's actual speed (to be set only if NAX = 0)
pos	REAL	Position regenerated (must be initialized to current position. Subsequently read-only)
vel	REAL	Speed regenerated (must be initialized to 0. Subsequently read-only)
acc	REAL	(Read only)

Examples

Example of use:

```
; definition of variable filMas of type STRU_MASTERF
STRU_MASTERF filMas
result = master_filter(filMas, 3)
velocityFiltered = filMas.vel
```

Structure STRU_MATCAM

This keyword allows a variable of type STRU_MATCAM (STRUCT_P 889) to be defined.

It is used by the mv_cam 425() function, which allows you to perform a third-order polynomial.

It consists of the following information:

Field	Type	Description
xInit	REAL	x cam beginning
yInit	REAL	y cam beginning
c	REAL	Coefficient c
b	REAL	Coefficient b
a	REAL	Coefficient a

Examples

Example declaration:

```
; definition of a moveCamma array of type STRU_MATCAM
STRU_MATCAM moveCamma[3]
```

Example of use:

© 2025 Robox SpA

```
; use of structure for a cam movement
; definition of a moveCamma array of type STRU_MATCAM
STRU_MATCAM moveCamma[3]
; definition of accessory variable for mv_cam
I32 segment_number = 3
; inserting the data into the structure
; (must be an array with a number
; of points equal to segment_number)
if(not(r(100)))
    moveCamma[0].xInit = r(220)
    moveCamma[0].yInit = r(221)
    moveCamma[0].c = r(222)
    moveCamma[0].b = r(223)
    moveCamma[0].a = r(224)
    moveCamma[1].xInit = r(230)
    moveCamma[1].yInit = r(231)
    moveCamma[1].c = r(232)
    moveCamma[1].b = r(233)
    moveCamma[1].a = r(234)
    moveCamma[2].xInit = r(240)
    moveCamma[2].yInit = r(241)
    moveCamma[2].c = r(242)
    moveCamma[2].b = r(243)
    moveCamma[2].a = r(244)
    r(100) = 1
endif
...
; cam execution
ip(2) = mv_cam(ip(1), segment_number, moveCamma)
```

Structure STRU_MODBUS

This keyword allows a variable of type STRU_MODBUS to be defined.

It is used by modbus management functions.

It consists of the following information:

Field	Type	Description
status	I32	Must be set = 0 for initialization. Output contains error code or 1 if waiting for completion of operation (such as the result of functions using this structure)
address	I32	Index of the first address to be read/write
quantity	I32	Number of address to be read/write

© 2025 Robox SpA

Field	Type	Description
endian	I32	Type of endian to be used: • EM_B_16 = 0 Big Endian in single word (default) • EM_L_16 = 1 Little Endian in single word • EM_B_32 = 2 Big Endian in 32 bit • EM_L_32 = 3 Little Endian in 32 bit • EM_MB_32 = 4 Middle Big Endian in 32 bit • EM_ML_32 = 5 Middle little Endian in 32 bit • EM_B_64 = 6 Big Endian in 64 bit • EM_L_64 = 7 Little Endian in 64 bit • EM_MB_64 = 8 Middle Big Endian in 64 bit • EM_ML_64 = 9 Middle little Endian in 64 bit
res_1	I32	trans_id
unitId	I32	unit_id
res_3	REAL	time_stamp

Examples

Example of use:

```
; definition of STRU_MODBUS
STRU_MODBUS myStru
INT valueRead[10]
INT idSlave = 0
INT return
...
; I prepare the support structure to execute the command.
; This is only needed the first time, then I always find it ready
; I want to read the holding registers from index 1 to index 10 and
insert the values into the value array
myStru.status = 0
myStru.address = 1
myStru.quantity = 10
myStru.unitId = idSlave
myStru.endian = EM_B_32
...
```

© 2025 Robox SpA

```
; I call the function to read the holding registers
; (mbc is the communication handle that is provided by the MCB_INIT 366
function)
return = mbc_read_hold_reg_s 379(mbc, myStru, valueRead)
; Note: The result of the function call is also available in the
field myStru.status
```

Structure STRU_MVGENITP

This keyword allows a variable of type STRU_MVGENITP (STRUCT_P 889) to be defined.

It is used by the mva_to_n_v 479() motion function.

It consists of the following information:

Field	Type	Description
nAx	I32	Axis number
target	REAL	Target to be reached (position). If the target is less than (pos_thr / 10), RTE considers the space null. So in case the user wishes to make micro advances (< pos_thr / 10) he will have to set the variable pos_thr appropriately
v	REAL	Speed to reach the target. **NOTE**: When interpolating multiple axes, the speed used may be less than the set speed (only the slowest axis to reach the target uses it, the others will use a lower speed).
vf	REAL	Final speed at target
accI	REAL	Initial maximum acceleration/deceleration to get to *v*
accF	REAL	Final maximum acceleration/deceleration to bring to target (on target I will have zero acceleration)
accH	REAL	Acceleration in case of hold

Examples

Example of declaration:

© 2025 Robox SpA

```
; definition of mov1 variable of type STRU_MVGENITP for the movement
of an axis
STRU_MVGENITP mov1
```

Example of a three-axis declaration:

```
; definition of mov_ax variable of type STRU_MVGENITP for the
movement of three axes
STRU_MVGENITP mov_ax[3]
```

Example of use:

```
; use of structure for a movement
; definition of variable m1 of type STRU_MVGENITP for the movement of
an axis
STRU_MVGENITP m1
; definition of accessory variables
I32 result
I32 restartFlag
; insertion of data into the structure
m1.nAx = 1 ; axis number
m1.target = rr(1000) ; desired position [units]
m1.v = rr(1001) ; steady state speed [units/sec]
m1.vf = rr(1002) ; final speed [units/sec]
m1.acci = rr(1003) ; initial acceleration [units/sec^2]
m1.accf = rr(1004) ; final acceleration [units/sec^2]
m1.accH = max_acc(1) ;deceleration value in case of hold
; restart flag
restartFlag = false
; I execute the movement
result = mva_to_n_v(restartFlag, m1 , 1, hold, fr)
; I check the result
if (result = M_REACHED)
    sr(1) = "I'm done!"
endif
```

NOTE: To access remember that indexes start from 0.

Structure STRU_MVITP

This keyword allows a variable of type STRU_MVITP (STRUCT_P) to be defined.

It is used by the axes movement function mva_to_n().

It consists of the following information:

Field	Type	Description
nAx	I32	Axis number
target	REAL	Position to be reached [units]
v	REAL	Speed [units/sec]

© 2025 Robox SpA

Field	Type	Description
acc	REAL	Acceleration [units/sec²]
dec	REAL	Deceleration [units/sec²]
decH	REAL	Deceleration at hold [units/sec²]

Examples

Example of single-axis definition:

```
; variable definition moveTrapezoidal of type STRU_MVITP (involves a
single axis)
STRU_MVITP moveTrapezoidal
```

Example of three-axis definition:

```
; variable moveTrapezoidal definition of type STRU_MVITP (involves
three axes)
STRU_MVITP moveTrapezoidal[3]
```

Example of thirty-two-axis definition:

```
; moveTrapezoidal variable definition of type STRU_MVITP (involves
thirty-two axes)
STRU_MVITP moveTrapezoidal[32]
```

Example of use:

```
; use of structure for a movement.
; variable definition m1 of type STRU_MVITP (involves a single axis)
STRU_MVITP m1
; definition of accessory variables
I32 result
I32 restartFlag
; insertion of data into the structure
m1.nAx = 1 ; axis number
m1.target = rr(1000) ; desired position [units]
m1.v = rr(1001) ; steady state speed [units/sec]
m1.acc = rr(1002) ; acceleration [units/sec^2]
m1.dec = rr(1003) ; deceleration [units/sec^2]
m1.decH = max_acc(1) ;value of deceleration in case of hold
; restart flag
restartFlag = false
; I execute the movement
result = mva_to_n471(restartFlag, m1 , 1, hold, fr)
; I check the result
if(result = M_REACHED)
    sr(1) = "I'm done!"
endif
```

NOTE: For access remember that indexes start from 0.

Structure STRU_MVITPCJ

This keyword allows a variable of type STRU_MVITPCJ (STRUCT_P 889) to be defined.

It is used by the mva_to_n_cj 475() motion function.

It consists of the following information:

Field	Type	Description
nAx	I32	Axis number
target	REAL	Position to be reached [units]
v	REAL	Speed [units/sec]
acc	REAL	Acceleration/deceleration [units/sec²]
jerk	REAL	Jerk [units/sec³]
decH	REAL	Deceleration on hold [units/sec²]

Examples

Example of a one-axis declaration:

```
; definition of variable moveJerkControlled of type STRU_MVITPCJ
STRU_MVITPCJ moveJerkControlled
```

Three-axis declaration example:

```
; definition of an array of three variables of type STRU_MVITPCJ
(three axes)
STRU_MVITPCJ mov[3]
```

Example of accessing member *v*:

```
; the v member of the first structure variable is accessed
mov[0].v
```

Example usage:

```
; use of structure for a mov
; definition of variable m1 of type STRU_MVITPCJ
STRU_MVITPCJ m1
; definition of accessory variables
I32 result
I32 restartFlag
; insertion of data into the structure
m1.nAx = 1 ; axis number
m1.target = rr(1000) ;  desired position [units]
m1.v = rr(1001) ; steady state speed [units/sec]
m1.acc = rr(1002) ; acceleration/deceleration [units/sec^2]
m1.jerk = rr(1003) ; jerk [units/sec^3]
```

© 2025 Robox SpA

```
m1.decH = max_acc(1) ;value of deceleration in case of hold
; restart flag
restartFlag = false
; I execute the movement
result = mva_to_n_cj(restartFlag, m1 , 1, hold, fr)
; I check the result
if (result = M_REACHED)
    sr(1) = "I'm done!"
endif
```

NOTE: For access remember that indexes start at 0.

Structure STRU_MVTO

This keyword allows a variable of type STRU_MVTO (STRUCT_P) to be defined.

It is used by the mv_to() motion function.

It consists of the following information:

Field	Type	Description
cPos	REAL	Current position [units]. To be updated the first time the function is called. In case the position of an axis is controlled, the assignment to be made is: stru_mvto.cPos = IP()

cSpeed	REAL	Current speed [units/sec].
		To be updated the first time the function is called.
		In case the position of an IP axis is controlled, the relative IV (speed) is automatically calculated by RTE. The assignment to be executed thus results:
		stru_mvto.cspeed = IV()
		In case the value of a register or variable is driven (thus not the IP), the calculation of *cSpeed* (current speed) must be done by the user.
		The simplest system is to calculate the rate of change of the controlled unit as follows:
		`; in rule_init`
		`oldValue = rr(1000)`
		`; in rule_prologue`

Field	Type	Description

		`velMeasure = (rr(1000)-oldValue) / si`
		`oldValue = rr(1000)`
		`; nelle rules 720 viene manipolato il valore della grandezza pilotata`
		`; prima della chiamata alla MV_TO`
		`stru_mvto.cpos = rr(1000)`
		`stru_mvto.cSpeed = velMeasure`
		`....`
		`rr(1000)= MV_TO 457 (........)`

dPos	REAL	Desired position [units]
dSpeed	REAL	Desired speed [units/sec]
acc	REAL	Acceleration [units/sec²]
dec	REAL	Deceleration [units/sec²]
delta	REAL	Time increment since the last execution of the function. If used in rules then it corresponds to the value of si. **NOTE**: It is not mandatory that the value refer to a time. Nothing prohibits it from being a space of a master axis, for example.

Examples

Example definition:

```
; definition of variable moveTrap of type STRU_MVTO
STRU_MVTO moveTrap
```

Example of use:

```
; use of structure for a moveTrap
; definition of variable m1 of type STRU_MVTO
STRU_MVTO m1
; definition of accessory variables
I32 result
; insertion of data into the structure
m1.cPos = ip(1) ; current position [units]
m1.cSpeed = iv(1) ; current speed [units/sec]
m1.dPos = rr(1002) ; desired position [units]
```

© 2025 Robox SpA

```
m1.dSpeed = rr(1003) ; desired speed [units/sec]
m1.acc = rr(1004) ; acceleration [units/sec^2]
m1.dec = rr(1005) ; deceleration [units/sec^2]
m1.delta = si ;  time increment
; I perform the motion
ip(1) = mv_to(result, m1 , 0)
; I check the result
if(result = M_REACHED)
   sr(1) = "I'm done!"
endif
```

NOTE: For access remember that indexes start at 0.

Structure STRU_MVTOCJ

This keyword allows a variable of type STRU_MVTOCJ (STRUCT_P) to be defined.

It is used by the mv_to_cj() motion function.

It consists of the following information:

Field	Type	Description
cPos	REAL	Current position [units]. To be updated the first time the function is called. In case the position of an axis is controlled, the assignment to be made is: stru_mvto.cPos = IP()

cSpeed	REAL	Current speed [units/sec].
		To be updated the first time the function is called.
		In case the position of an IP axis is controlled, the relative IV (speed) is automatically calculated by RTE. The assignment to be executed thus results:
		stru_mvto.cspeed = IV()
		In case the value of a register or variable is driven (thus not the IP), the calculation of cSpeed (current speed) must be done by the user.
		The simplest system is to calculate the rate of change of the driven variable as follows:
		`; in rule_init`
		`oldValue = rr(1000)`

Field	Type	Description

		`; in rule_prologue` 732
		`velGrandezza = (rr(1000)-oldValue) / si`
		`oldValue = rr(1000)`
		`; nelle rules` 720 `viene manipolato il valore della grandezza pilotata`
		`; prima della chiamata alla MV_TO`
		`stru_mvto.cpos = rr(1000)`
		`stru_mvto.cSpeed = velGrandezza`
		`....`
		`rr(1000)= MV_TO_CJ` 461
		`(........)`

cAcc	REAL	Current acceleration [units/sec²]. To be updated the first time the function is called. In case the position of an IP axis is controlled, the corresponding IA (acceleration) is automatically calculated by RTE. The assignment to be executed thus results: stru_mvto.cAcc = IA() Otherwise, see *cSpeed.*
dPos	REAL	Desired position [units]
dSpeed	REAL	Desired speed [units/sec]
dAcc	REAL	Desired acceleration [units/sec²]
dJerk	REAL	Jerk [units/sec³]
delta	REAL	Time increment since the last execution of the function. If used in rules then it corresponds to the value of si. It is not mandatory that the value refer to a time. Nothing prohibits it from being a space of a master axis, for example.

Examples

Example declaration:

© 2025 Robox SpA

```
; definition of variable stk of type STRU_MVTOCJ
STRU_MVTOCJ stk
```

Example of usage.

```
; use of structure for a movement
; definition of variable m1 of type STRU_MVTOCJ
STRU_MVTOCJ m1
; definition of accessory variables
I32 result
I32 restartFlag
; insertion of data into the structure
m1.cPos = ip(1) ; current position [units]
m1.cSpeed = iv(1) ; current speed [units/sec]
m1.cAcc = ia(1) ;  current acceleration [units/sec^2]
m1.dPos = rr(1002) ;  desired position [units]
m1.dSpeed = rr(1003) ;  desired speed [units/sec]
m1.dAcc = rr(1004) ; desired  acceleration [units/sec^2]
m1.dJerk = rr(1005) ; jerk [units/sec^3]
m1.delta = si ;time increment
; I perform the movement
ip(1) = mv_to_cj(result, restartFlag, m1 , hold, fr)
; I check the result
if (result = M_REACHED)
    sr(1) = "I'm done!"
endif
```

Structure STRU_MVTOCJV

This keyword allows a variable of type STRU_MVTOCJV (STRUCT_P) to be defined.

It is used by the mv_to_cjv() motion statement.

It consists of the following information:

Field	Type	Description
cPos	REAL	Current position [units]. To be updated the first time the function is called. In case the position of an axis is controlled, the assignment to be made is: stru_mvto.cPos = IP()
cSpeed	REAL	Current speed [units/sec]. To be updated the first time the function is called. In case the position of an IP axis is controlled, its IV (speed) is automatically calculated by

Field	Type	Description

		RTE. The assignment to be executed thus results:
		stru_mvto.cspeed = IV()
		In case the value of a register or variable is controlled (thus not the IP), the calculation of *cSpeed* (current speed) must be done by the user.
		The simplest system is to calculate the rate of change of the controlled variable as follows:
		`; in rule_init` 730
		`oldValue = rr(1000)`
		`; in rule_prologue` 732
		`velGrandezza = (rr(1000)-oldValue) / si`
		`oldValue = rr(1000)`
		`; in rules` 720 `the value of the controlled magnitude is manipulated ; before the call to MV_TO`
		`stru_mvto.cpos = rr(1000)`
		`stru_mvto.cSpeed = velValue`
		`....`
		`rr(1000)= MV_TO_CJV` 461
		`(........)`

cAcc	REAL	Current acceleration [units/sec²]. To be updated the first time the function is called. In case the position of an IP axis is controlled, the corresponding IA (acceleration) is automatically calculated by RTE. The assignment to be executed thus results: stru_mvto.cAcc = IA() Otherwise, see *cSpeed*
dPos	REAL	Desired position [units]
dSpeed	REAL	Desired speed [units/sec]
dAcc	REAL	Acceleration [units/sec²]

© 2025 Robox SpA

Field	Type	Description
dJerk	REAL	Jerk [units/sec³]
delta	REAL	Time increment since the last execution of the function. If used in rules then it corresponds to the value of si. It is not mandatory that the value refer to a time. Nothing prohibits it from being a space of a master axis, for example.

Examples

Example declaration:

```
; definition of variable stk of type STRU_MVTOCJV
STRU_MVTOCJV stk
```

Example of use:

```
; use of structure for a movement
; definition of variable m1 of type STRU_MVTOCJV (involves a single
axis)
STRU_MVTOCJV m1
; definition of accessory variables
I32 result
; insertion of data into the structure
m1.cPos = ip(1) ;  current position [units]
m1.cSpeed = iv(1) ; current speed [units/sec]
m1.cAcc = ia(1) ;  current acceleration [units/sec^2]
m1.dPos = rr(1002) ; desired position [units]
m1.dSpeed = rr(1003) ;  desired speed [units/sec]
m1.dAcc = rr(1004) ;desired  acceleration [units/sec^2]
m1.dJerk = rr(1005) ; jerk [units/sec^3]
m1.delta = si ;time increment
; I execute the movement
result = mv_to_cjv 461(m1 , hold, fr)
; I set the returned values of cPos, cSpeed and cAcc
set_kinematics 569(1, m1.cPos, m1.cSpeed, m1.cAcc)
; check the result
if (result = M_REACHED)
    sr(1) = "I'm done!"
endif
```

Structure STRU_MVTOCJV_INFO

This keyword allows a variable of type STRU_MVTOCJV_INFO (STRUCT_P 889) to be defined.

It is used by the motion function mv_to_cjv_info 463().

It consists of the following information:

Field	Type	Description
targetSpeed	REAL	Speed target value [units/sec]
dAcc	REAL	Acceleration value to reach target [units/sec²]
dJerk	REAL	Jerk value to reach target [units/sec³]
cPos	REAL	Current position [units]. To be updated the first time the function is called
cSpeed	REAL	Current speed [units/sec]. To be updated the first time the function is called
cAcc	REAL	Current acceleration [units/sec³]. To be updated the first time the function is called
posInfo	REAL	Position reached [units]
timeInfo	REAL	Time required [sec]

Examples

Example declaration:

```
; definition of variable mvinfo of type STRU_MVTOCJ_INFO
STRU_MVTOCJV_INFO mvinfo
```

Example of use:

```
; use of structure
; definition of variable m1 of type STRU_MVTOCJ_INFO
STRU_MVTOCJV_INFO m1
; definition of accessory variables
I32 result
; insertion of data into the structure
m1.TargetSpeed = rr(1000) ;  target speed value [units/sec]
m1.dAcc = rr(1001) ;  desired acceleration [units/sec^2]
m1.dJerk = rr(1002) ;  jerk [units/sec^3]
m1.cPos = ip(1) ;  current position [units]
m1.cSpeed = iv(1) ;  current speed [units/sec]
m1.cAcc = ia(1) ;  current acceleration [units/sec^2]

; I perform the movement
result = mv_to_cjv_info 463 (m1)
; I check the result
; if it is ok, I store the values of position reached and time needed
```

© 2025 Robox SpA

```
if(result)
    sr(1) = "I'm done!"
    rr(1100) = m1.PosInfo ;position reached
    rr(1101) = m1.TimeInfo ;time needed
endif
```

Structure STRU_MVTOCJVE

This keyword allows a variable of type STRU_MVTOCJVE (STRUCT_P 889) to be defined.

It is used by the mv_to_cjv 461() motion function.

It consists of the following information:

Field	Type	Description
stopPosition	REAL	(read only) If the field *dSpeed* is set to zero, the stop position is automatically written
stopTime	REAL	(read only) In case of imposing the field *dSpeed* to zero, the relative time required for stopping is automatically written
cPos	REAL	Current position [units]. To be updated the first time the function is called. In case the position of an axis is controlled, the assignment to be made is: stru_mvto.cPos = IP()
cSpeed	REAL	Current speed [units/sec]. To be updated the first time the function is called. In case the position of an IP axis is controlled, its IV (speed) is automatically calculated by RTE. The assignment to be executed thus results: stru_mvto.cspeed = IV() In case the value of a register or variable is controlled (thus not the IP), the calculation of *cSpeed* (current speed) must be done by the user. The simplest system is to calculate the rate of change of the controlled variable as follows:

© 2025 Robox SpA

Field	Type	Description

		`; in rule_init` 730
		`oldValue = rr(1000)`
		`; in rule_prologue` 732
		`velValue = (rr(1000)-oldValue) / si`
		`oldValue = rr(1000)`
		`; in rules` 720 `the value of the controlled magnitude is manipulated ; before the call to MV_TO`
		`stru_mvto.cpos = rr(1000)`
		`stru_mvto.cSpeed = velValue`
		`....`
		`rr(1000)= MV_TO_CJV` 461
		`(........)`

cAcc	REAL	Current acceleration [units/sec²]. To be updated the first time the function is called. In case the position of an IP axis is controlled, the corresponding IA (acceleration) is automatically calculated by RTE. The assignment to be executed thus results: stru_mvto.cAcc = IA()
dPos	REAL	Desired position [units]
dSpeed	REAL	Desired speed [units/sec]
dAcc	REAL	Acceleration [units/sec²]
dJerk	REAL	Jerk [units/sec³]
delta	REAL	Time increment since the last execution of the function. If used in rules then it corresponds to the value of si. It is not mandatory that the value refer to a time. Nothing prohibits it from being a space of a master axis, for example

Examples

Example of definition

```
; definition of variable stk i type STRU_MVTOCJVE
STRU_MVTOCJVE stk
```

Example usage:

```
; use of structure for a movement
; definition of variable m1 type STRU_MVTOCJVE
STRU_MVTOCJVE m1
; definition of accessory variables
I32 result
; data entry into the structure
m1.cPos = ip(1) ; current position [units]
m1.cSpeed = iv(1) ;  current speed [units/sec]
m1.cAcc = ia(1) ;  current acceleration [units/sec^2]
m1.dPos = rr(1002) ;  desired position [units]
m1.dSpeed = rr(1003) ;  desired speed [units/sec]
m1.dAcc = rr(1004) ;desired  acceleration [units/sec^2]
m1.dJerk = rr(1005) ;  jerk [units/sec^3]
m1.delta = si ;time increment
; I execute the movement
result = mv_to_cjv(m1, hold, fr)
; I set the returned values of cPos, cSpeed and cAcc
set_kinematics(1, m1.cPos, m1.cSpeed, m1.cAcc)
; check the result
if(result = M_REACHED)
    sr(1) = "I'm done!"
else
    if(m1.dSpeed = 0) ; in this case dSpeed was set to zero
        rr(1100) = m1.stopPosition ;  then I store the values of
m1.stopPosition
        rr(1101) = m1.stopTime ; and of m1.stopTime
    endif
endif
```

Structure STRU_MVZC

This keyword allows a variable of type STRU_MVZC (STRUCT_P) to be defined.

It is used by the mva_zc() function, which executes the axis zero cycle.

It consists of the following information:

Field	Type	Description
nAx	I32	Axis number
preset	REAL	Preset quota [units]
velPos	REAL	Speed for preliminary movement to actual cycle0 [units/sec].

© 2025 Robox SpA

Field	Type	Description
		NOTE: If selected zero cycle to end of strokes, it represents error epos(*nAx*) in units to consider end of strokes reached.
velZc	REAL	Speed for final movement during cycle0 [units/sec]
acc	REAL	Acceleration [units/sec²]
jerk	REAL	Jerk [units/sec³]
spSch	REAL	Movement space on final movement [units]
filter	REAL	Time filter in seconds to filter out bounces of the micro (the micro must remain in the same state for the specified time for it to be assumed valid). **NOTE**: If zero cycle to end of strokes is selected and you have drive in CANopen/EtherCAT present in the transducer selection list of the project configurator, it represents the torque reduction value in percent [0-:-100].
zcFlag.0 [bit 0]	U32	Speed sign: With homing without micro (*zcFlag*.3=1) or monodirectional (*zcFlag*.4=1): • 0, negative speed • 1, positive speed With homing using micro (*zcFlag*.3=0): • 0, negative speed if micro state in 1, positive if micro state in 0 • 1, positive speed if micro state in 1, negative if micro state in 0
zcFlag.1 [bit 1]	U32	Micro switching edge selection: With homing using micro only (*zcFlag*.3=0, *zcFlag*.2=0) the micro's edge is the *preset* quota assignment point:

© 2025 Robox SpA

Field	Type	Description
		• 0, the falling edge of the micro • 1, the rising edge of the micro With homing using micro + full-precision (*zcFlag*.3=0, *zcFlag*.2=1) the *preset* quota assignment point is the one identified by the full-precision with the micro in the state indicated by this bit: • 0, the logic low state of the micro • 1, the logic high state of the micro With homing that do not use micro (*zcFlag*.3=1) it is not considered
zcFlag.2 [bit 2]	U32	Enable/disable full-precision: • 0, disable full precision • 1, enable full precision **NOTE**: Full-precision is the maximum possible precision that can be obtained from the axis transducer, and whether it can be used depends on the type of transducer itself. Examples: • the CH0 if the transducer is enc_inc type • the motor electric revolution quota if Fieldbus type transducer (if the device is managed and if it is available)
zcFlag.3 [bit 3]	U32	Micro usage selection: • 0, enable micro • 1, disable micro (**NOTE**: In this case full-precision must be enabled (*zcFlag*.2=1))
zcFlag.4 [bit 4]	U32	Selection of the homing direction:

© 2025 Robox SpA

Field	Type	Description
		• 0, execution of the homing is bidirectional • 1, the execution of the homing is monodirectional
zcFlag.5 [bit 5]	U32	Enable the limitation of space during preliminary movements between eos (max_str and min_str): • 0, the space during preliminary handling is unlimited • 1, the space during the preliminary movement is limited by the value (max_str(axis)-min_str(axis)) In case this value is exceeded it will be reported diagnostic (#1)
zcFlag.6 [bit 6]	U32	If bidirectional movement (*zcFlag*.4=0) or unused micro (*zcFlag*.3=1) is ignored. If monodirectional movement (*zcFlag*.4=1) and micro used (*zcFlag*.3=0) is used to enable/disable preliminary movements (micro variation search) (#1) • 0, enabled • 1, disabled
zcFlag.7 [bit 7]	U32	Selection of cycle0 to end of strokes (#1) • 0, disabled • 1, enabled
zcFlag.8 [bit 8]	U32	ON_POSITION zero cycle selection. The current position of the axis becomes the indicated preset value. RTE also sets the relevant bit in the predefined variable c0_done
zcFlag.9 [bit 9]	U32	Flag enabling repositioning to the preset quota. The speed used is the speed specified in the *velZc* parameter. This bit cannot be set to 1 if bit 10 is also set to 1. If bits 9 and

© 2025 Robox SpA

Field	Type	Description
		10 are both set to 1 alarm 9900 will be generated. **NOTE**: When monodirectional zero cycles are selected, this flag is ignored.
zcFlag.10 [bit 10]	U32	Flag to enable repositioning within the software end of strokes(min_str and max_str). In case the preset quota is outside the software end of strokes and *zcFlag*.10 = 1 RTE provides movement of the axis to the nearest end of stroke. The axis is stopped (with ramping) as soon as RTE detects that its quota is inside the end of strokes. The speed used is that indicated in the *velPos* parameter. This bit cannot be set to 1 if bit 9 is also set to 1. If bits 9 and 10 are both set to 1 the alarm 9900 will be generated
zcFlag.11 [bit 11]	U32	The bit is used only if a monodirectional homing is selected, with the zero micro and search of the initial variation. In this case if the bit is in 1, when the zero micro variation is detected the speed *velPos* is kept for a space equal to the parameter *spSch*. Then speed *velZc* is activated to perform the final path. Actually the initial positioning phase is prolonged of *spSch*
zcFlag.12 [bit 12]	U32	It is used only for cycle of 0 to end of strokes. Enables, at the time of the crash the stopping of the evolution of the ideal quota in a controlled way with controlled return of the ip to the crash position
zcFlag.13 [bit 13]	U32	It is used only for cycle of 0 to end of strokes. Disables speed control (abs 133(ip-cp) > zc.*velPos*) when searching for end of strokes (#2)
zcFlag.14 [bit 14]	U32	From RTE 34.28.0

© 2025 Robox SpA

Field	Type	Description
		It is used to realign RAWIP and RAWCP. It is used only in 'On Position' type homing with position controlled digital drives to realign the offset used for RAWIP calculation. The homing cycle must be performed in 'missing power'. Put another way, it realigns in 'missing power' (with open position loop) the RAWIP with the RAWCP

(#1) Available from RTE versions v32.x.x for INTEL processors, v33.3.3 for PPC processors

(#2) Available from RTE version v34.24.10

Legenda

- Ciclo0/zero cycle/homing: is the final mission of the movements performed by the mva_zc, or the one containing the assignment of the preselection quote to the axis CP
- Preliminary movement to zero cycle: this defines all that series of movements (preliminary steps) possibly performed by the mva_zc that precede zero cycle

Examples

Example declaration for an axis:

```
; definition of home variable of type STRU_MVZC for the movement of
an axis.
STRU_MVZC home
```

Example declaration for three axes:

```
; definition of home variable of type STRU_MVZC for the movement of
three axes
STRU_MVZC home[3]
```

Structure STRU_PHADJ

This keyword allows a variable of type STRU_PHADJ(STRUCT_P 889) to be defined.

It is used by the mv_phase_adj 448() motion function.

It consists of the following information:

Field	Type	Description
error	REAL	It is the deviation of the slave axis from the desired phase position (positive if you need to accelerate to reach the phase, negative if you need to slow down).

© 2025 Robox SpA

Field	Type	Description
		Once the function is working, a new correction can be executed by changing the error value and also resetting the space run by the master *deltaMaster* runtime (e.g., on the rising edge of an input)
masterCycle	REAL	It is the master axis space available to perform phase correction
velMinSlave	REAL	This is the lower limit of speed allowed for the movement of the slave axis. In case the correction formula requires going below the minimum speed, the value returned by the function would be *velMinSlave*. In case of intervention of the limitation, the correction of the phase error will result only partial. In the case of a negative value, it is limited to 0
velMaxSlave	REAL	This is the upper speed limit allowed for slave axis movement. In case the correction formula requires going above the maximum speed, the value returned by the function would be equal to *velMaxSlave*. In case of limitation, the correction of the phase error will result only partial
accMaxSlave	REAL	This is the maximum acceleration value accepted for the slave axis. The amount of correctable error is automatically reduced so as not to exceed the set acceleration value. The error parameter provided by the user is automatically changed by the function
vRef	REAL	This is the base speed of the slave axis, i.e., the speed to be held in the absence of corrections. This parameter must be updated by the application

© 2025 Robox SpA

Field	Type	Description
		software before each call to the function
deltaMaster	REAL	This is the elapsed master space since the start of the function call. The correction occurs in the range of *deltaMaster* between zero and *masterCycle*. For *deltaMaster* values greater than *masterCycle*, the function returns the base speed *vRef* and return code M_SYNCHRONIZED This parameter must be updated by the application software before each call to the function

Examples

Esempio di definizione:

```
; definition of nastro1 variable of type STRU_PHADJ
STRU_PHADJ nastro1
```

Esempio di utilizzo:

```
; use of structure for a movement
; definition of variable m1 of type STRU_PHADJ
STRU_PHADJ m1
; definition of auxiliary variables
I32 esito
I32 fase
I32 time
I32 posMasterIn
I32 VelAxSlave
; data entry into the structure
; first execution
if (first_time 321())
    m1.masterCycle = rr(1001)                          ;master axis
space [units]
    m1.velMinSlave = rr(1002)                          ;minimum
slave speed [units/sec]
    m1.velMaxSlave = rr(1003)                          ;maximum
slave speed [units/sec]
    m1.accMaxSlave = rr(1004)                          ;maximum
slave acceleration [units/sec^2]
endif

; I perform operations based on the variable "fase"
```

© 2025 Robox SpA

```
select (fase)
    case 0                                          ;initial phase
(0)
        m1.error = rr(1000)                         ;slave axis
offset
        posMasterIn = ip(1)                         ;I store the
initial position of the master axis
        fase = 1                                    ;I move on to the
next stage
    case 1                                          ;phase 1
        m1.vRef = iv(1)                             ;slave base speed
[units/sec]
                                                    ;in this case is
set equal
                                                    ;to the speed of
the master axis
        m1.deltaMaster = ip(1) - posMasterIn        ;delta master
[units]
                                                    ;is set equal to
the difference between the current position
                                                    ;and the initial
position of the master axis
        VelAxSlave = mv_phase_adj(esito, m1)           ;I run the
function
        iv(2) = VelAxSlave                          ;I assign the
calculated velocity to the physical axis
        if (esito = M_SYNCHRONIZED)                 ;if I am
synchronized I go to phase 2
            fase = 2
            time = TFB                              ;I memorize the
instant of time
        endif
        break
    case 2                                          ;phase 2
        _if ((TFB - time) > 1)                      ;wait a second
            fase = 0                                ;then I return to
phase 0
        break

end_select
```

Structure STRU_PHADJ2

Validity: **RC3E v33.5.8-7** (or later)

This keyword allows a variable of type STRU_PHADJ2 (STRUCT_P) to be defined.

It is used by the mv_phase_adj2() motion function.

It consists of the following information:

<table>
<tr><th>Field</th><th>Type</th><th>Description</th></tr>
<tr><td>corrType</td><td>I32</td><td>Type of correction profile:
Comparison table of correction profiles taking the triangular profile as reference:
<table>
<tr><th></th><th>corrType</th><th>Vmax</th><th>Amax</th><th>Jerk max</th></tr>
<tr><td>0</td><td>Triangular</td><td>100</td><td>100</td><td>infinity in 3 points</td></tr>
<tr><td>1</td><td>Parabolic</td><td>75</td><td>150</td><td>infinity in 2 points</td></tr>
<tr><td>2</td><td>Sinusoidal</td><td>78.5</td><td>123.25</td><td>infinity in 2 points</td></tr>
<tr><td>3</td><td>Jerked</td><td>100</td><td>200</td><td>100</td></tr>
<tr><td>4</td><td>Raised cosine</td><td>100</td><td>157</td><td>123</td></tr>
<tr><td>5</td><td>Pentic</td><td>100</td><td>150</td><td>150</td></tr>
</table></td></tr>
<tr><td>error</td><td>REAL</td><td>It is the deviation of the slave axis from the desired phase position (positive if you need to accelerate to reach the phase, negative if you need to slow down).
Once the function is working, a new correction is executed by changing the error value and also resetting the space traveled by the deltaMaster runtime (e.g., on the rising edge of an input)</td></tr>
<tr><td>masterCycle</td><td>REAL</td><td>It is the master axis space available to perform phase correction</td></tr>
<tr><td>velMinSlave</td><td>REAL</td><td>This is the lower limit of speed allowed for the movement of the slave axis. In case the correction formula requires going below the minimum speed, the value returned by the function would be velMinSlave.
In case of limitation, the phase error correction will result only partial, and the state variable will take the value M_SPEED_LIMITED.</td></tr>
</table>

© 2025 Robox SpA

Field	Type	Description
		In the case of a negative value it will be limited to 0
velMaxSlave	REAL	This is the upper speed limit allowed for slave axis movement. In case the correction formula requires going above the maximum speed, the value returned by the function would be equal to *velMaxSlave*. In case of limitation, the phase error correction will result only partial, and the state variable will take value M_SPEED_LIMITED
accMaxSlave	REAL	This is the maximum acceleration value accepted for the slave axis. If the limitation is triggered, the phase error correction will result only partial, and the state variable will take value M_ACCEL_LIMITED
vRef	REAL	This is the base speed of the slave axis, i.e., the speed to be held in the absence of corrections. This parameter must be updated by the application software before each call to the function
deltaMaster	REAL	This is the elapsed master space since the start of the function call. The correction occurs in the range of *deltaMaster* between zero and *masterCycle*. For *deltaMaster* values greater than *masterCycle*, the function returns the base speed *vRef* and return code M_SYNCHRONIZED This parameter must be updated by the application software before each call to the function
vRefMaster	REAL	Master target speed (if not specified leave 0 or IV(master))
vSlave	REAL	Current slave speed This parameter must be updated by the application software before each call to the function
si	REAL	Sampling interval
kTrap	REAL	Minor base/major base ratio trapezoid (possibly recalculated at

© 2025 Robox SpA

Field	Type	Description
		start of correction and reset at end of correction)
camValue	REAL	Current correction cam value (read only). Varies between 0 and 1

Examples

Example declaration:

```
; definition of variable tape1 of type STRU_PHADJ
STRU_PHADJ2 tape1
```

Example of use in rule:

```
RULE R_AUTO_2
axes(AX_2)

MOTION
 if (first_time 321())
 myHook.corrType = 0 ; triangle
 myHook.masterCycle = rrPassoMaster ; master space available to
retrieve error
 myHook.velMinSlave = 0 ; minimum slave axis speed during error
recovery
 myHook.velMaxSlave = MAX_SPE(AX_2) ; maximum slave axis speed during
error recovery
 myHook.accMaxSlave = MAX_ACC(AX_2) ;  maximum slave axis
acceleration during recovery error
 myHook.si = SI
 myHook.error = 0
 mv_status = 0
 endif
 if(rise 536(inp(1))) ; little red ADV button
 myHook.vRefMaster = rrVelMaster ; master steady state speed (e.g.
mv_ramp 451(master, rrVelMaster , rrAccMaster))
 myHook.error = slave_error
 mv_status = M_INITIALIZING
 posMasterInitial = IP(AX_1)
 endif
 ;
 ; speed that the slave axis should hold in the absence of
corrections
 ;
 myHook.vRef = iv(AX_1)
 ;
 ; current speed of the slave
 ;
 myHook.vSlave = iv(AX_2)
 ;
 ; space traveled by the master from the beginning of the correction
```

© 2025 Robox SpA

```
 ;
 myHook.deltaMaster = ip(AX_1) - posMasterInitial
 ;
 ; I move the axis
 ;
 iv(AX_2) = mv_phase_adj2 448(mv_status, myHook)

 if(rise 536(mv_status = M_SYNCHRONIZED))
 ; correction finished
 endif

END_MOTION

END_RULE
```

Structure STRU_PM_PD

Validity: **RC3E v33.9.1-9** (or later)

This keyword allows a variable of type STRU_PM_PD to be defined.

It is used by the axioline communication function axio_pm_r 160().

It consists of the following information:

Field	Type	Description
L1_voltage	I32	Voltage in mV on phase L1 (*)(***)
L2_voltage	I32	Voltage in mV on phase L2 (*)(***)
L3_voltage	I32	Voltage in mV on phase L3 (*)(***)
L1_current	I32	Current in mA on phase L1 (*)
L2_current	I32	Current in mA on phase L2 (*)
L3_current	I32	Current in mA on phase L3 (*)
NeutralConductor_current	I32	Current in mA on neutral (*)
L1_Real_PE	I32	Active power/energy in W/Wh on phase L1 (**)
L2_Real_PE	I32	Active power/energy in W/Wh on phase L2 (**)
L3_Real_PE	I32	Active power/energy in W/Wh on phase L3 (**)

© 2025 Robox SpA

Field	Type	Description
L1_Reactive_PE	I32	Power/Reactive energy in VAr/VArh on phase L1 (**)
L2_Reactive_PE	I32	Power/Reactive energy in VAr/VArh on phase L2 (**)
L3_Reactive_PE	I32	Power/Reactive energy in VAr/VArh on phase L3 (**)
L1_Apparent_PE	I32	Apparent power/energy in VA/VAh on phase L1 (**)
L2_Apparent_PE	I32	Apparent power/energy in VA/VAh on phase L2 (**)
L3_Apparent_PE	I32	Apparent power/energy in VA/VAh on phase L3 (**)

Note: (*) If in object 0x0E11 is saved the value 0x0 (default) then in these fields there will be the mean square value, if instead the value 0x1 is saved the instantaneous value.

Note: (**) If in object 0x0E12 is saved the value 0x0 (default) then in these fields there will be the power, if instead the value 0x1 is saved the energy.

Note: (***) If in object 0x0E13 is saved the value 0x0 (default) then in these fields there will be line voltage, if instead the value 0x1 is saved the phase-to-phase voltage.

Examples

Example of use

```
INT pm_result
INT rid, slot ; rid and slot of device AX1_SE_IOL4
STRU_PM_PD pm ; buffer used to store PM process data

; get now the information from the module
pm_result = axio_pm_r 160(rid, slot, pm, sizeof(pm));
if (pm_result < 0)
 ; write here your diagnostic
else
 ; test if all the lines have a voltage bigger than 100V
 if ((pm.L1_voltage > 100000 ) and (pm.L2_voltage > 100000 ))
 ;do something
 endif
endif
```

Structure STRU_REACHTARGET

This keyword allows a variable of type STRU_REACHTARGET (STRUCT_P 889) to be defined.

It is used by the mv_reach_target 451() motion function.

It consists of the following information:

Field	Type	Description
velSlave	REAL	Current slave speed, must be updated by application software
velMinSlave	REAL	Minimum speed allowed during docking
velMaxSlave	REAL	Maximum speed allowed during docking
accMAxSlave	REAL	Maximum acceleration allowed during docking
vTarget	REAL	Desired speed of the slave at the synchronism position
ds	REAL	Missing space at the slave at the synchronism point
tm	REAL	Missing time to the master at the synchronism point
si	REAL	Sample interval

Examples

Example declaration:

```
; definition of variable mov1 of type STRU_REACH_TARGET
STRU_REACHTARGET mov1
```

Example of use:

```
; use of structure for a mov
; definition of variable m1 of type STRU_REACH_TARGET
STRU_REACHTARGET m1
; definition of accessory variables
I32 result
I32 phase
REAL targetMaster
REAL targetSlave
REAL rrPassoMaster
REAL rrPassoSlave
REAL rrAccSlave
; data entry into the structure
; first execution
if(first_time())
 m1.velMinSlave = rr(1002) ; minimum slave speed [units/sec]
 m1.velMaxSlave = rr(1003) ; maximum slave speed [units/sec]
 m1.accMaxSlave = rr(1004) ; maximum slave acceleration [units/sec^2]
 m1.si =  si ;sampling time
 phase = 0
endif
; I perform operations based on the variable "phase"
```

© 2025 Robox SpA

```
select (phase)
 case 0 ;=== initial phase ===
 ; with the activation of input 100 I start tracking
 if (rise(536)(inp(100)))
   iv(2) = 0
   targetMaster = ip(1) + rrPassoMaster
   targetSlave = ip(2) + rrPassoSlave
   step = 1
 endif
 break
 case 1 ; ===  phase 1 ===
 ; Calculate time until the master reaches the target quota
 m1.tm = (targetMaster - ip(1)) / iv(1)
 ; Calculate space that is missing for the slave
 m1.ds = targetSlave - ip(2)
 ; I update speed of the master
 m1.vtarget = iv(1)
 ; I update speed of the slave
 m1.velSlave = iv(2)
 ; I move the axis
 iv(2) = mv_reach_target(451) (result, m1)
 ; if I have reached the target I move to step 2
 if (result = M_REACHED)
   phase = 2
 endif
 break
 case 2 ; ===  phase 2 ===
 ; I continue tree to the master for 100 units.
 ; then I move on to phase 3
 iv(2) = iv(1)
 _if ((ip(ax2) - targetSlave) > 100 )
   phase = 3
 break
 case 3 ; ===  phase 3 ===
 ; stop the slave axis
 iv(2) = ramp(530)(iv(2) , 0 , rrAccSlave)
 ; then return to phase 0
 if ((iv(2) < K_EPSILON))
   phase = 0
 endif
 break
end_select
```

Structure STRU_SAFETY_PRJINFO

Validity: **RC3E v33.9.1-3** (or later)

This keyword allows a variable of type STRU_SAFETY_PRJINFO to be defined.

It is used by the safe_prj_info(549)() function.

It consists of the following information:

Field	Type	Description
desc	STRING	Customer description
prjName	STRING	Project name
ver	STRING	Version
ct	STRING	Date and time compiled
isle	U8	Island number
reserved_0	U8	Reserved
HB_crc	U32	Project header CRC
LB_crc	U32	Logical block CRC
AB_crc	U32	Address block CRC

Examples

Example of use:

```
INT result_spi
STRU_SAFETY_PRJINFO spi

result_spi = safe_prj_info(1, spi)
if (result_spi < 0)
 ; write here your diagnostic
else
 ; set value in registers for HMI
 SR(10) = spi.prjName
 R(10) = spi.HB_crc
 R(11) = spi.LB_crc
 R(12) = spi.AB_crc
endif
```

Structure STRU_SOE_IDN

Validity: **RC3E v33.5.19-3** (or later)

This keyword allows a variable of type STRU_SOE_IDN to be defined.

This variable is used by the soe_ridn() and soe_widn() functions to read/write SOE parameters.

It consists of the following information:

Field	Type	Description
rid	I32	Robox ID associated with the device
idn	I32	Number of the S or P parameter to be read or written

© 2025 Robox SpA

Field	Type	Description
element	I32	Only if read - element to be read 0 is assumed as DATA () (see table element codes below)
cmd	I32	Only if write - management by commands - 0 (not command) 1,2,3 (see code table) If read set to 0
isp	I32	Parameter type selection : • if 0 you manage S-0-idn • if 1 it manages P-0-idn
datatype	I32	Type of data to read/write (see code table)
datalen	I32	If used for SOE_RIDN: maximum size in bytes of the area to store the response If used for SOE_WIDN: size in bytes of the data to be written.
phase	I32	Working phase/execution mode: • 1, initialization pass-through execution (**) • -1, blocking execution initialization(**) • 2, reserved RTE • 3, reserved RTE • 4, reserved RTE • 5, reserved RTE • 6, operation successfully executed • 0, uninitialized or error • other, illegal value Only values marked with (**) ARE THE ONLY ones that can be written into this variable by the user. The blocking value (-1) is changed by the instruction, in case of consecutive executions of the SOE_RIDN 577 or SOE_WIDN 589 functions the value -1 MUST be reset before execution of the same

Field	Type	Description
nric	I32	Number of characters received. Useful in case of arrays to deduce how many elements were received
res_1	I32	Reserved
res_2	I32	Reserved
res_3	I32[260]	Reserved

Table element codes:

	Element	Code
STS	parameter status	0x01
NAME	parameter name	0x02
ATTR	parameter attribute	0x04
UNIT	parameter units	0x08
MIN	parameter minimum value	0x10
MAX	parameter maximum value	0x20
DATA	parameter data (Default)	0x40
DEF	parameter default value	0x80

Code Table:

	Code
1	Executed command procedure (with autocheck if ended)
2	Stop command procedure
3	Send the specified value to handle command procedure

Data type codes (dataType)

Code	Data type
0	Automatic selection (default)
2	INTEGER8
3	INTEGER16
4	INTEGER32
5	UNSIGNED8

© 2025 Robox SpA

Code	Data type
6	UNSIGNED16
7	UNSIGNED32
8	REAL32
9	VISIBLE_STRING
10	OCTET_STRING
17	REAL64
21	INTEGER64
27	UNSIGNED64
45	BITARR8
46	BITARR16
47	BITARR32
-2	ARRAY INTEGER8
-3	ARRAY INTEGER16
-4	ARRAY INTEGER32
-5	ARRAY UNSIGNED8
-6	ARRAY UNSIGNED16
-7	ARRAY UNSIGNED32
-8	ARRAY REAL32
-9	VISIBLE_STRING SOE

Examples

Example definition:

```
; definition of variable s of type STRU_SOE_IDN
STRU_SOE_IDN s
```

Structure STRU_STAT

This keyword allows a variable of type STRU_STAT (STRUCT_P 889) to be defined.

It is used by the statistics 595() function.

It consists of the following information:

Field	Type	Description
initdone	I32	Flag to store initialization done
min	REAL	Minimum value reached by the variable on which the statistic is performed
max	REAL	Maximum value reached by the variable on which the statistic is performed
average	REAL	Average value of the variable on which the statistic is performed
ncount	REAL	Desired number of samples on which to do average
actcount	REAL	Current number of samples (modulated on *ncount*)
acttot	REAL	Totalizer sum of sample values

Examples

Example declaration:

```
; definition of variable myStatistics of type STRU_STAT
STRU_STAT myStatistics
```

Example of use:

```
; use of data contained in the structure
; definition of myStatistics variable of type STRU_STAT
STRU_STAT statT1
; definition of auxiliary variables
REAL returns, T1, minVal, maxVal
; set to 1000 the number of cycles over which to average
statT1.ncount = 1000
...
; in the main loop
; I measure the duration of the loop
T1 = loop_time 363()
; I store the average value in the variable in "returns"
returns = statistics 595(T1,statT1)
; I store the minimum value in the variable in "minVal"
minVal = statT1.min
; I store the maximum value in the variable in "maxVal"
maxVal = statT1.max
```

© 2025 Robox SpA

Structure STRU_SYNCHRO

This keyword allows a variable of type STRU_SYNCHRO (STRUCT_P 889) to be defined.

It is used by the mv_synchro 454() motion function.

It consists of the following information:

Field	Type	Description
qMaster	REAL	Master position. Must be updated before each function call
velMaster	REAL	Speed of the master. Must be updated before each function call
qSyncMaster	REAL	Master axis synchronism quota
qSlave	REAL	Position of the slave. Must be updated before each function call
velSlave	REAL	Slave speed. Must be updated before each function call
qSyncSlave	REAL	Synchronism quota of the slave axis
velSyncSlave	REAL	Speed that the slave must have at the synchronism quota *qSyncSlave*
velMaxSlave	REAL	This is the upper speed limit allowed for slave axis movement. In case the correction formula required to rise above the maximum speed, the value returned by the function would be equal to *velMaxSlave*
accMaxSlave	REAL	This is the maximum acceleration value accepted for the slave axis. The amount of correctable error is automatically reduced so that the set acceleration value is not exceeded
si	REAL	Sampling time si
kappa	REAL	Allows the trajectory of the slave to be warped: the higher the value, the more the slave movement is anticipated, i.e., more space is attempted during the first half of the mission.

© 2025 Robox SpA

Field	Type	Description
		Reasonable values range from 1.0 (no advance) up to 3.0 (maximum advance)

Examples

Example declaration:

```
; definition of variable m1 of type STRU_SYNCHRO
STRU_SYNCHRO m1
```

Example

```
; use of structure for a movement
; definition of variable m1 of type STRU_SYNCHRO
STRU_SYNCHRO m1
; definition of accessory variables
I32 result
I32 step
REAL rrPassoMaster
REAL rrPassoSlave
REAL rrAccSlave
; data entry into the structure
; first execution
if(first_time321())
 m1.velMaxSlave = rr(1003) ; maximum slave speed [units/sec]
 m1.accMaxSlave = rr(1004) ; maximum slave acceleration [units/sec^2]
 m1.si = si ; sampling time
 m1.kappa = rr(1006) ; trajectory deformation parameter [1--3]
 phase = 0
endif
; I perform operations based on the variable "phase"
select (phase)
 case 0 ;initial phase (0)
 ; I set master docking quota.
 ; When the master will reach m1.qsyncMaster also.
 ; the slave will have reached m1.qsyncSlave and at that
 ; point it will have the speed m1.velsyncslave
 m1.qsyncMaster = ip(1) + rrPassoMaster
 ; set slave docking quota
 m1.qsyncSlave = ip(2) + rrPassoSlave
 step = 1
 break
 case 1 ;phase 1
 ; I update master parameters
 m1.qMaster = ip(1)
 m1.velMaster = iv(1)
 ; I update slave parameters
 m1.qSlave = ip(2)
 m1.velSlave = iv(2)
```

© 2025 Robox SpA

```
 ; I set the slave speed on the docking point
 m1.velsyncslave = iv(1)
 ; I move the axis
  ip(2) = mv_synchro(result, m1)
 ; If I have achieved synchronization I move to step 2
 if(result = M_REACHED)
 phase = 2
 endif
 break
 case 2 ;phase 2
 ; I continue tree to the master for 100 units
 ; then I move on to phase 3
 iv(2) = iv(1)
 _if ((ip(2) - m1.qsyncSlave) > 100)
 phase = 3
 break
 case 3 ;phase 3
 ; stop the slave axis
  iv(2) = ramp(iv(2) , 0 , rrAccSlave)
 ; then I switch to phase 4
 if ((iv(2) < K_EPSILON))
 phase = 4
 time = TFB
 endif
 break
 case 4 ;phase 4
 ; I wait for 1 second
 _if ((tfb - time) > 1)
 ; Then I go back to phase 0
 phase = 0
 break
end_select
```

Structure STRU_TRACK

Validity: **RC3E v33.3.2** (or later)

This keyword allows a variable of type STRU_TRACK (STRUCT_P) to be defined.

It is used by the tracking() function.

It consists of the following information:

Field	Type	Description
despos	REAL	Desired quota to be reached. can also be a quantity that evolves over time (quota of another axis)
desspeed	REAL	It must be updated each time the function is called and represents the velocity of the

© 2025 Robox SpA

Field	Type	Description
		despos field. In case of an axis it represents the IV of the axis itself while in case of a variable it can be calculated as the variation of the field *despos* over time: [new target - previous *despos* / field *delta*]
actpos	REAL	Current quota of the unit controlled by the function
actspeed	REAL	Current speed of the unit controlled by the function
maxspeed	REAL	Maximum speed of the unit controlled by the function
acc	REAL	Maximum acceleration of the unit controlled by the function
posmin	REAL	Lower slave end of strokes of the unit controlled by the function
posmax	REAL	Upper slave end of strokes of the unit controlled by the function
delta	REAL	Sampling interval (si)

Examples

Example declaration:

```
; definition of trk variable of type STRU_TRACK
STRU_TRACK trk
```

Example usage:

```
; use of the structure for a movement
; if the tracking command is active, axis 2 tracks axis 1
; definition of variable m1 of type STRU_TRACK
STRU_TRACK m1
; definition of accessory variables
I32 trk
I32 returned
; if the tracking command is given.
if(trk)
 ; I insert the data into the structure
 m1.delta = si; sampling interval [sec]
 m1.desspeed = iv(1) ; desired speed [unit/sec]
 m1.despos = ip(1)   ; desired position [unit]
 m1.actpos = ip(2)   ;  current position [unit]
```

© 2025 Robox SpA

```
 m1.actspeed = iv(2) ; current speed [unit/sec]
 m1.maxspeed = max_spe(2) ; maximum speed [unit/sec]
 m1.acc = max_acc(2)    ;  maximum acceleration [unit/sec^2]
 m1.posmin = min_str(2) ;lower end of strokes [unit]
 m1.posmax = max_str(2) ;upper end of strokes [unit]
 ; I call the tracking function
  returns = tracking(m1)
 ;I assign the calculated velocity to the physical axis
 ip(2) = m1.actpos
 ; ; if the tracking command is not active instead...
else
 ; ; ...stop the axis
 iv(2) = ramp(iv(2) ,0 , max_acc(2))
endif
```

Structure STRUCT

This keyword allows you to define data structures.

The definition of the struct begins with the keyword "struct" followed by the name that the user wishes to assign to the struct.

Then it is possible to declare the type and name of the fields that make up the struct.

Finally, the struct is terminated with the keyword "end_struct".

Examples

Below is a struct named "CUSTOM_DATE" containing three integers named "day", "month", "year" respectively:

```
struct CUSTOM_DATE
 INT day
 INT month
 INT year
end_struct
```

Below we initialize the previous struct with the foundation date of ROBOX:

```
CUSTOM_DATE foundation

foundation.day = 20
foundation.month = 2 ;february
foundation.year = 1975
```

Example of using vectors in a struct:

```
struct MYSTRUCT
 INT intNumber[32] ;  32 integer
 FLOAT floatingPointNumber[16] ; 16 float
end_struct
...
MYSTRUCT testStruct
testStruct.intNumber[5] = 1975
```

© 2025 Robox SpA

```
testStruct.floatingPointNumber[3] = 3,1415926536
```

Example of using struct vectors:

```
struct MYSTRUCT
 INT intNumber
 FLOAT floatingPointNumber
end_struct
...
MYSTRUCT testStruct[8] ; declare an array of 8 structs
testStruct[0].intNumber = 1975
testStruct[4].floatingPointNumber = 3,1415926536
;example of using Fields in a calculation
testStruct[6].floatingPointNumber = testStruct[0].intNumber *
testStruct[4].floatingPointNumber
```

NOTE: You can't use registers to index arrays of structures or internal index arrays.

Structure STRUCT_P

It is equivalent to STRUCT 888 with the difference that the data are byte-aligned.

NOTE: Typical use for defining structures for exchanging data with external devices.

Structure TABSTR

Validity: **Rule**, **Task**

This keyword allows the definition of a STRING CONSTANTS table.

The definition is closed by the keyword ENDTAB.

String constants may be grouped into tables. An identifier must be associated with the table; the individual string will be identified by the table identifier to which the index number of the string in question must be associated. The index of the first string is 0, that of the second is 1, and so on.

Field	Type
"text"	STRING
"text"	STRING
"text"	STRING
"text"	STRING
"text"	STRING
.....	STRING

Examples

Example of use:

© 2025 Robox SpA

```
; definition of a STRING variable named message
STRING message
; definition of a TABSTR structure named commands.
tabstr commands
 "Speed"
 "Position"
 "Acceleration"
endtab
; message will contain "the speed is excessive"
message = "the " # commands[0] # " is excessive"
; message will contain "the acceleration is excessive"
message = "the " # commands[2] # " is excessive"
```

Type BOOL

This keyword allows you to define a boolean variable.

Syntax	**BOOL**
Representable values	1 (TRUE), 0 (FALSE)
Note	

Examples

Example of use:

```
;----------------------------------------
; definition of local variables task 1
;----------------------------------------
BOOL pippo, pluto
pippo = FALSE
pluto = 1
```

Type CAST

The term "forcing" (CAST) refers to the ability to convert one unit of a certain type into another of the specified type. The quantity in question can be indifferently a constant, a variable or the result of a mathematical expression.

Field	Type	Description
Forcing to I8	I8	expression/variable/constant
Forcing to I16	I16	expression/variable/constant
Forcing to I32	I32	expression/variable/constant
Forcing to U8	U8	expression/variable/constant
Forcing to U16	U16	expression/variable/constant

© 2025 Robox SpA

Field	Type	Description
Forcing to U32 901	U32	expression/variable/constant
Forcing to REAL 898	REAL	expression/variable/constant
Forcing to FLOAT 892	FLOAT	expression/variable/constant
Forcing to INT 894	Equivalent to I32	
Forcing to UNSIGNED 901	Equivalent to U32	

NOTE: A forcing should be used whenever it is desired to change the normal sequence of conversions implicitly made within a mathematical expression.

Examples

Example of use:

```
; The contents of real register 4 is converted to integer
; then 2 is added to it and assigned to r(5).
r(5) = (I32)(rr(4)) + 2

; The integer constant 2 is converted (automatically) to type REAL
; and is then added to the value contained in real register 4.
; The result is then converted to type I32 and assigned to r(5).
; Note that in this example the forcing (I32) is superfluous
; since it does not alter the sequence of operations that would have
occurred in its absence.
; In contrast, in the previous example the forcing(I32)
; produces a sequence of events different from the natural one.
r(5) = (I32)(rr(4) + 2)

; The contents of the integer register r(2) is converted to real
; and is then multiplied by the real constant 65536.0
; The content of the integer register r(1) is converted to U32
; and subsequently converted to real (implicit conversion)
; and then added to the result of the previous multiplication.
; The final result is assigned to real register 5.
rr(5) =(U32)(r(1)) + (REAL)(r(2)) * 65536.0
```

Type CHAR

It is equivalent to I8 895.

Validity: **Rule**, **Task**

This keyword allows you to define a variable containing a character, with reference to ASCII encoding.

Syntax	**CHAR**
Minimum representable number	-128

© 2025 Robox SpA

Syntax	**CHAR**
Maximum representable number	127
Size	1 Byte
Note	If precedes, in brackets, the variable name performs a force operation CAST 890

Type COUNTER

This keyword allows a variable of type COUNTER to be defined.

This variable is used by the counter management instructions ct_cu 672() , ct_cd 670() , ct_cur 673() , ct_cdr 671().

Examples

Example statement:

```
; definition of the variables counter1 and counter2 of type COUNTER.
COUNTER counter1, counter2
```

Type FLOAT

This keyword allows the type of variable to be defined as floating point (4 byte format - IEEE Std 754 and 854).

Syntax	**FLOAT**
Minimum representable number	8.43e-37
Maximum representable number	3.37e+38
Size	4 Byte
Note	If precedes, in parentheses, the variable name performs a force operation CAST 890

Examples

Example usage:

```
;----------------------------------------
; local variable definition task 1
;----------------------------------------
FLOAT pippo, pluto
FLOAT support
; example of forcing type "CAST"

FLOAT f1, f2, f3, f4
...
r(100) = 1000000000 ; 1e9
```

© 2025 Robox SpA

```
r(102) = 5
r(103) = 128
r(104) = 5
...
; multiplication takes place between integers.... so with the above
data we get out of the validity range of the INT
; 5 * 1e9 = 1000000000 -> 0x12A05F200, which is truncated to
0x2A05F200 -> 705032704
f1 = r(100) * r(101) ; f1 = 7.050327e+8
; the multiplication is between integers but the cast (FLOAT) forces
the conversion from INT to FLOAT before performing the
multiplication.
f2 = (FLOAT)r(100) * r(101) ; f2 = 5e+9
if (R(103))
 ; division takes place between integers
 f3 = R(102) / r(103) ; f3 = 25
 ; data is converted to FLOAT before division is done
 f4 = (FLOAT)r(102) / r(103) ; f3 = 25.6
else
 f3 = 0
 f4 = 0
endif
```

Type I16

Validity: **Rule**, **Task**

This keyword allows you to define a 16-bit signed integer numeric variable in two's complement (C2).

Syntax	**I16**
Minimum representable number	-32768
Maximum representable number	32767
Size	2 Byte
Note	If precedes, in brackets, the variable name performs a force operation CAST 890

Examples

Some numbers in the two representation systems:

Hexadecimal representation	Decimal representation
0XA	10
0AH	10

© 2025 Robox SpA

Hexadecimal representation	Decimal representation
10H	16
0FFH	255
0x200	512
0xFE00	-512
7FFFH	32767
8000H	-32768
0FFFFH	-1

Example of use:

```
;----------------------------------------
; definition of local variables task 1
;----------------------------------------
I16 pippo, pluto
I16 support
; example of force type "CAST"
; appo is an integer type variable with 32-bit sign
; pippo is an integer type variable with 16 bit sign
I32 appo
I16 peppo
; the value of appo is converted from 32 to 16 bits
; then assigned to peppo
peppo = (I16) (appo)
```

Type I32

Validity: **Rule**, **Task**

These keywords allow a 32-bit integer numeric variable to be defined in signed two's complement.

NOTE: The two keywords are equivalent. The use of I32 or INT is left to the taste of the programmer.

Syntax	**I32** **INT**
Minimum representable number	-2147483648
Maximum representable number	2147483647
Size	4 Byte
Note	If either keyword precedes, in brackets, the name of a variable, a CAST 890 forcing operation is performed

Examples

Some numbers in the two representation systems:

Hexadecimal representation	Decimal representation
0XA	10
0AH	10
10H	16
0FFH	255
0FFFFH	65535
7FFFFFFFH	2147483647
80000000H	-2147483648
0FFFFFFH	-1

Example of use:

```
;----------------------------------------
; definition of local variables of task 1
;----------------------------------------
I32 pippo, pluto
I32 support
INT integerVar_1
; example of forcing type "CAST"
; given two real variables (whose absolute value content is less than
1073741824) to get the sum of the integer parts
; one can proceed as follows :
REAL first, second, sumResult, integerSumResult
...
first = 1.23445
second = 3.98765
...
sumResult = first + second ; sumResult = 5.22215
integerSumResult = (I32)first + (I32)second ; integerSumResult = 4
```

Type I8

Validity: **Rule**, **Task**

This keyword allows you to define an 8-bit integer numeric variable in 2's complement with sign.

Syntax	**I8**
Minimum representable number	-128
Maximum representable number	127

© 2025 Robox SpA

Syntax	**I8**
Size	1 Byte
Note	If precedes, in brackets, the variable name performs a force operation CAST 890

Examples

Some numbers in the two representation systems:

Hexadecimal representation	Decimal representation
0XA	10
0AH	10
10H	16
7FH	127
80H	-128
0FFH	-1

Example of use:

```
;----------------------------------------
; definition of local variables task 1
;----------------------------------------
I8 pippo, pluto
I8 support
; example of force type "CAST"
; appo is an integer type variable with 16 bit sign
; pippo is an integer type variable with 8-bit sign
I16 appo
I8 peppo
; the value of appo is converted from 16 to 8 bits
; then assigned to peppo
peppo = (I8)(appo)
```

Type POWER_SET

This keyword allows the variable type to be defined as POWER_SET.

WARNING: When a Power Set is created via RDE environment, a variable of type POWER_SET with the name of the Power Set is automatically defined (within the cfgdefs.i3 file). Normally, therefore, there is no need to define the variable by task R3.

Syntax	**POWER_SET**

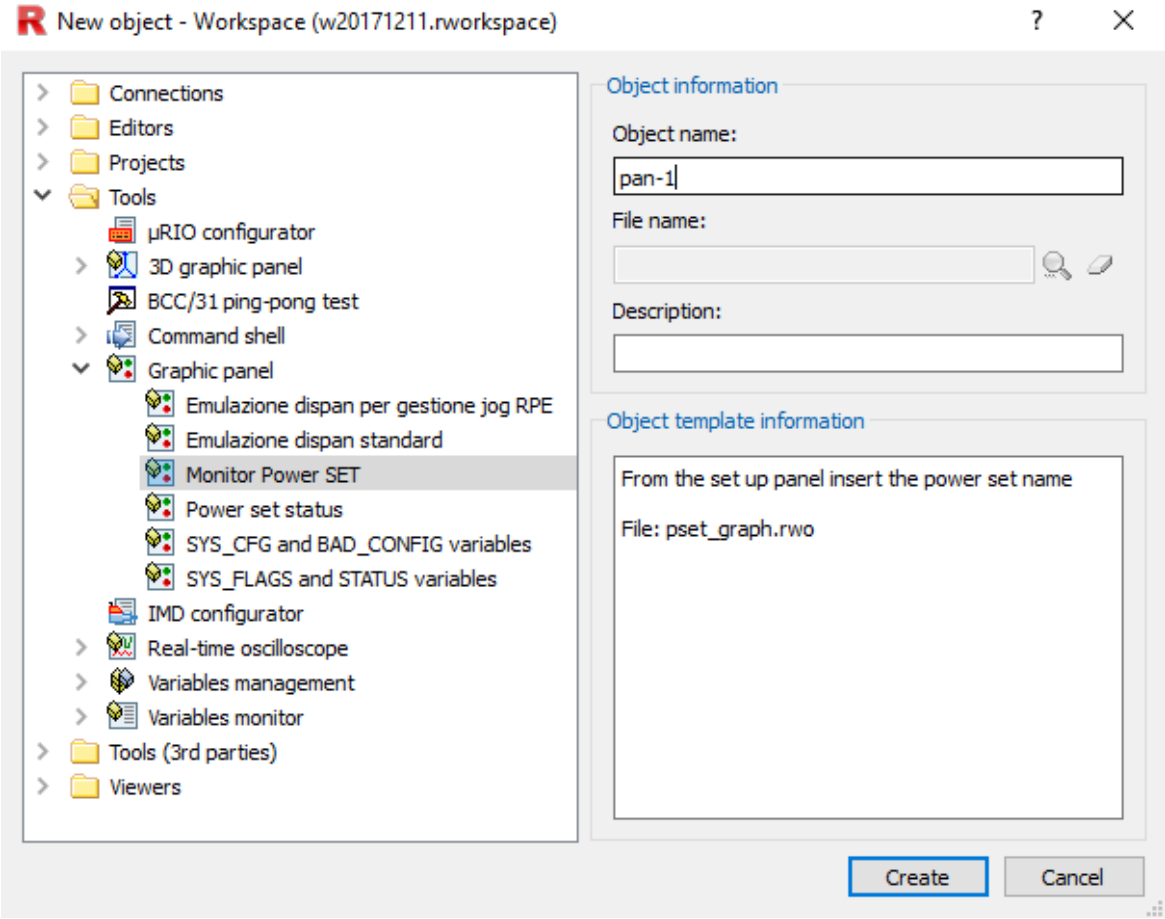
It consists of the following information:

Field	Type	R/W	Description
power_enable	BOOL	R/W	Enabling the power set
channel_enable	U32	R/W	Enabling the power set drives
channel_enable_feedback	U32	R	Mask of the the feedback signals of individual drives
channel_reset_fault	U32	R/W	Resetting an alarm of one or more drives
drive_reset	U32	R/W	Reset of one or more drives
status	U32	R	Power set status information (see ps_status 517)
channel_enabled	U32	R	Information about the enabled drives of the power set
channel_fault	U32	R	Information about the faulted drives of the power set
eba	BOOL	R	Consent to energization of the power set by alarms
prq	BOOL	R	Presence of the power request signal
gpfb	BOOL	R	Presence of the feedback signal series
gpfm	U32	R	Mask of the status of individual feedback signals defined by configurator
gpfc	U32	R	Mask of the state of individual feedback signals of CAN402 managed axes
pam	U32	R	Physical Axis Mask
lam	U32	R	Logic Axis Mask

It is possible to access power set information either by means of variables or by means of special functions.

NOTE: There is a default graphical panel in RDE in which the information in the POWER_SET structure is displayed. To bind it to the desired power set, you must enter the power set name as the value of PSNAME, which can be found under settings -> parameters

© 2025 Robox SpA



Examples

Example usage:

```
; definition (found in cfgdefs.i3) of a variable of type POWER_SET
POWER_SET isle1
; access to power set information via variable
; the variable "status_pwr" will contain the status information of
the power set "isle1"
U32 state_pwr
status_pwr = isle1.status
```

Type REAL

Validity: **Rule**, **Task**

This keyword allows the variable type to be defined as REAL, i.e. floating point 8 Byte (double).

Syntax	**REAL**
Minimum representable number	2.225074e-308

Syntax	**REAL**
Maximum representable number	1.797693e+308
Size	8 Bytes
Note	If it precedes the variable name in brackets it performs a force operation CAST 890 You can initialize the value of the variable using the following syntax: REAL variableName = { 0.0 }

Examples

Example of use:

```
;----------------------------------------
; definition of local variables task 1
;----------------------------------------
REAL pippo, pluto
REAL support
; two-element array definitions with associated initialization
REAL pull overX[2] = { 0.0, 0.0 }
REAL pull overY[2] = { 0.0, 50.0 }
REAL pull overZ[2] = { 50.0, 50.0 }
; example of forcing type "CAST"
; appo is a variable of type integer with 32-bit sign
; peppo is a variable of type REAL
I32 894 appo
REAL peppo
; the value of appo is converted from 32 bits to REAL
; then assigned to peppo
peppo = (REAL) (appo)
```

Type STRING

Validity: **Rule**, **Task**

This keyword allows the type of variable to be defined as STRING, that is, strings of characters.

Syntax	**STRING [(len=128)]**
len	String length (optional, default 128 (127+1))
Note	The terminator is 0 and occupies the position following the last character

The terminator is managed by RTE.

Examples

© 2025 Robox SpA

Example definition:

```
; with the following declaration we associate:
; - to the string pippo a maximum length of 32 characters (31 usable,
plus terminator 0)
; - to the string zippo a maximum default length of 128 characters
(127 usable, plus terminator 0)
; - to string string1 a maximum length of 2000 characters (1999
usable, plus terminator 0)
; If the length is specified, the number of characters in it
; shall not exceed said value, but may be less than or equal to it.
STRING pippo(32), zippo, string1(2000)
```

Type TIMER

This keyword allows a variable of type TIMER to be defined.

This variable is used by the Timer management instructions tm_on 754() , tm_of 753() , tm_int 751() , tm_sqw 756().

Examples

Example declaration:

```
; definition of the variables timer1 and timer2 of type TIMER.
TIMER timer1, timer2
```

Type U16

This keyword allows a 16-bit unsigned integer numeric variable to be defined.

Syntax	**U16**
Minimum representable number	0
Maximum representable number	65535
Size	2 Byte
Note	If precedes, in brackets, the variable name performs a force operation CAST 890

Examples

Some numbers in the two representation systems:

Hexadecimal representation	Decimal representation
0XA	10
0AH	10

© 2025 Robox SpA

Hexadecimal representation	Decimal representation
10H	16
7FH	127
80H	128
0FFH	255
7FFFH	32767
8000H	32768
0FFFFH	65535

Example of use:

```
;----------------------------------------
; definition of local variables task 1
;----------------------------------------
U16 pippo, pluto
U16 support
; example of force type "CAST"
; appo is a 32-bit unsigned integer type variable
; pippo is a 16-bit unsigned integer type variable
U32 appo
U16 peppo
; the value of appo is converted from 32 to 16 bits
; then assigned to peppo
peppo = (U16)(appo)
```

Type U32

This keyword allows a 32-bit unsigned integer numeric variable to be defined.

Syntax	**U32**
Minimum representable number	0
Maximum representable number	4294967295
Size	4 Byte
Note	If precedes, in brackets, the variable name performs a force operation CAST 890

Examples

Some numbers in the two representation systems:

Hexadecimal representation	Decimal representation
0XA	10

© 2025 Robox SpA

Hexadecimal representation	Decimal representation
0AH	10
10H	16
7FH	127
80H	128
0FFH	255
7FFFH	32767
8000H	32768
0FFFFH	65535
7FFFFFFFH	2147483647
80000000H	2147483648
0FFFFFFH	4294967295

Example of use:

```
;----------------------------------------
; definition of local variables task 1
;----------------------------------------
U32 pippo, pluto
U32 support
; example of forcing type "CAST"
; given two real variables (whose absolute value content is less than
1073741824) to get the sum of the integer parts
; one can proceed as follows :
REAL first, second, sumResult, integerSumResult
...
first = 1.23445
second = 3.98765
...
sumResult = first + second ; sumResult = 5.22215
integerSumResult = (U32)first + (U32)second ; integerSumResult = 4
```

Type U8

This keyword allows an unsigned 8-bit integer numeric variable to be defined.

Syntax	**U8**
Minimum representable number	0
Maximum representable number	255
Size	1 Byte

© 2025 Robox SpA

Syntax	**U8**
Note	If precedes, in brackets, the variable name performs a force operation CAST 890

Examples

Some numbers in the two representation systems:

Hexadecimal representation	Decimal representation
0XA	10
0AH	10
10H	16
7FH	127
80H	128
0FFH	255

Example of use:

```
;----------------------------------------
; definition of local variables task 1
;----------------------------------------
U8 pippo, pluto
U8 support
; example of force type "CAST"
; appo is a 16-bit unsigned integer type variable
; pippo is an 8-bit unsigned integer type variable
U16 appo
U8 peppo
; the value of appo is converted from 16 to 8 bits
; then assigned to peppo
peppo = (U8)(appo)
```

Miscellaneous

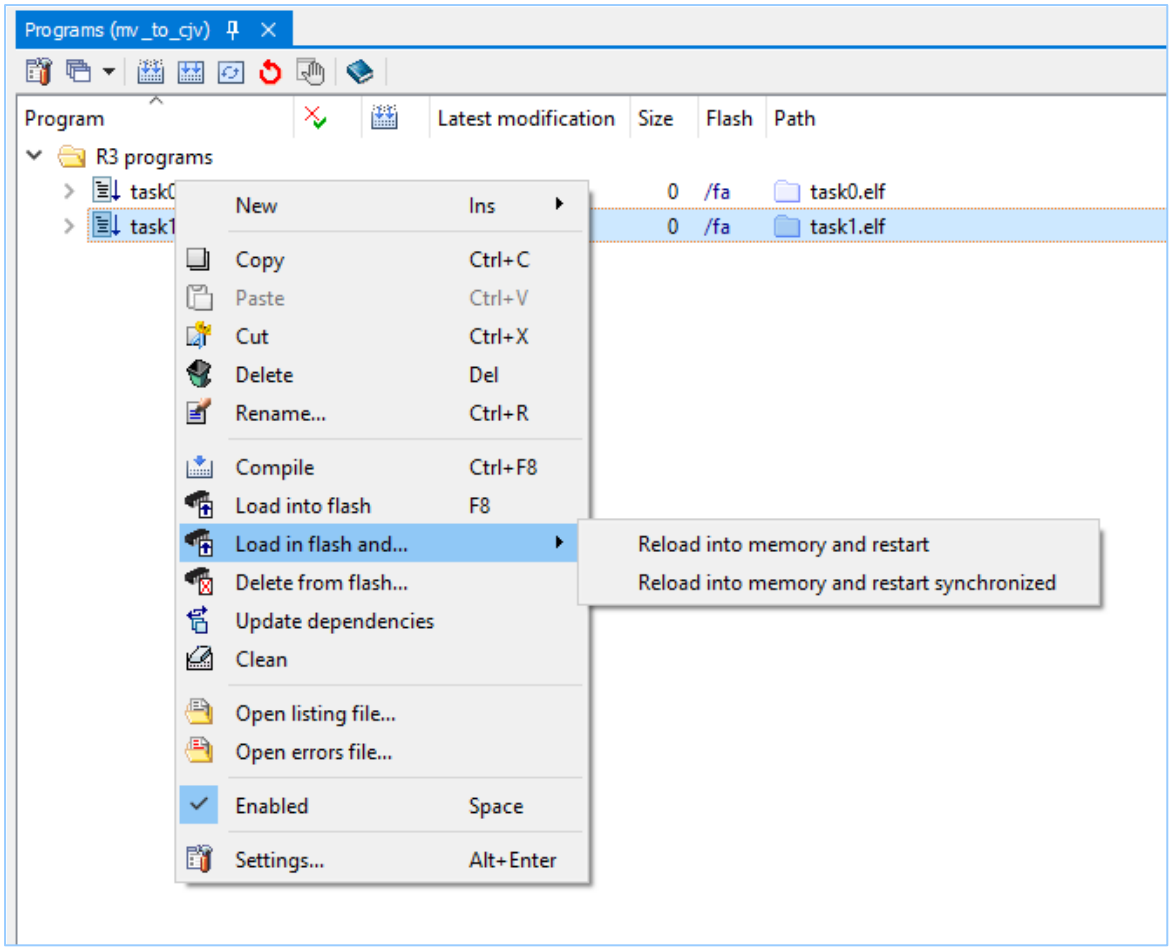
LIVE Changes

From RTE v33.18, referring to programs written in R3, it is possible to "inject" modified programs while the control is in run mode (orange mode led flashing) without having to perform a system boot.

Operation

After making the modify to the program and compiling it with zero errors, from the project window it is possible to perform the operation in two different ways at the choice of the programmer:

© 2025 Robox SpA



The programmer will then have to structure his program (task/rule) in such a way as to avoid re-executing initialization functions that are typically put before the main loop. The default variables RESTARTED and SYNC_RESTART can be used for this purpose.

The volatile registers R, RR, and SR that are reset at boot are not reset in the case of a "LIVE" change.

The RESTART_POINT 719 keyword allows selection of the point at which the program should stop, before starting (from the beginning) the loaded one (this only if synchronized restart has been selected)

The keyword RESTART_POINT can be used multiple times in the same program.

Reload into memory and restart

- The program (task) is stopped at the instruction in which it is and is replaced with the new one
- The predefined variable RESTARTED signals the presence of this situation
- If the change is made to rules the system must be in missing power (GLOBAL_POWER = false)

Reload into memory and restart synchronized

- The program (task) is stopped at the first RESTART_POINT instruction and is replaced with the new one
- The predefined variable SYNC_RESTART signals the presence of this situation

© 2025 Robox SpA

- The keyword RESTART_POINT can be used multiple times in the same program
- If the change is made to rules the system must be in missing power (GLOBAL_POWER = false)

Examples

Example of use:

```
; boot backing already executed
lit rInitDone r(1000)
lit rOperativeMode r(1001)

; if first boot init is started so I can handle the LIVE
modifications
_if (not rInitDone)
    call InitTask1
rInitDone = true

; Main loop
__main_loop__
    ; stop point in the running program
    restart_point

    ; management of operating mode changes
    call HndModeChange
    out(150) = inp(1)

    ; machine state manager
    select (rOperativeMode)
        case M_MIS_POWER         ; Missing power handling
           call HndMisPower
           break
        ...
    end_select
    ...
end_main_loop
```

© 2025 Robox SpA

Index

- - -

- 776

- # -

781

- $ -

$check_array 770
$close_loop 770
$define 772
$else 772
$endif 772
$error 770
$expand 770
$ifdef 772
$ifndef 772
$include 770, 773
$rule 770
$rule_inp 770
$rule_periodic 770
$rule_time 770
$step 770
$task 770
$undef 770, 774
$use_rpe 770
$warning 770
$write_on_extern 770

- * -

* 776

- / -

/ 776

- _ -

__main_loop__ 650
_else 651
_if 651

- + -

+ 776

- < -

< 784
<= 784
<> 784

- = -

= 781, 784

- > -

> 784
>= 784

- A -

abs 133
abs_mod 134
acos 134
ah_cmd 135
ah_get_d 136
ah_get_s 137
ah_info 139
ah_log 140
ah_log_us 141
al2am 142
alarm_reset 145
alarm_reset_par 146
alarm_reset_par_us 149
alarm_set 151
alarm_set_us 152
Alarms handling 92
Alarms management 92
and 782
Arithmetical operators 776
as_get_d 154
asin 155
Assignment operators 781
atan 156
atan2 156
aux 652
axes 653
axes_m 654
axio_iol_r 157
axio_iol_w 159
axio_pm_r 160
axio_robj 162
axio_wobj 177
Axioline handling 94
Axioline Management 94

- B -

b<n> 191
b0 191, 654
b1 192, 654

© 2025 Robox SpA

b2 193, 655
b3 194, 655
b4 194, 655
b5 195, 655
b6 195, 656
b7 196, 656
BCC Communication handling 99
BCC Communication Management 99
bcc_get_double 197
bcc_get_float 197
bcc_get_i16 198
bcc_get_i32 198
bcc_get_i8 199
bcc_get_string 199
bcc_get_tcp_clients_info 200
bcc_get_u16 201
bcc_get_u32 201
bcc_get_u8 202
bcc_h_setup 202
BCC_MSGUSER 793
bcc_receive 204
bcc_reply_ack 204
bcc_reply_msg 205
bcc_reply_nack 206
bcc_send 206
bcc_set_double 656
bcc_set_float 657
bcc_set_i16 658
bcc_set_i32 658
bcc_set_i8 659
bcc_set_string 659
bcc_set_u16 660
bcc_set_u32 660
bcc_set_u8 661
Binary operators (bit) 777
bit_mask_add 661, 770
bit_mask_remove 663
bit_mask_set 665, 770
BOOL 890
break 667

- C -

call 667
CANopen/EtherCAT handling 96
CANopen/EtherCAT Management 96
case 669
CAST 890
check_mv_follow2 207
check_mva_follow2 210
check_mva_follow2_m 213
CNET handling 99
CNET Management 99
co_asw_sdo 215
co_controlword_om 218
co_controlword_state 220
co_get_ws_diagno 221
co_par_download 224
co_par_upload 227
co_rerr 231
co_rnmt 232
co_robj 234
co_rpdo 241
co_send_sdo 242
co_statusword 246
co_sync 247
co_wnmt 248
co_wobj 249
co_wpdo 254
coe_robj 255
coe_wobj 259
command 264
cond_val 265
Constants 17
continue 670
cos 265
COUNTER 892
Counter handling 109
Counter management 109
crc16_be 266
crc16_le 268
ct_cd 670
ct_cdr 671
ct_cu 672
ct_cur 673
ct_cv 269, 674
ct_load 674
ct_pv 270, 675
ct_qd 270
ct_qu 271
ct_reset 675

© 2025 Robox SpA

- D -

Data handling 110
Data Management 110
Data types 787
dec 271
default 675
delta_enc16 272
delta_enc32 273
diff 274
do 676
dw<n> 275
dw0 275, 676
dw1 277, 677
dwbe2le 277
dwbe2lef 278
dwbe2mbe 278
dwbe2mle 279
dwell 677
dwle2be 279
dwle2bef 280
dwle2mbe 280
dwle2mbef 281
dwle2mle 281
dwle2mlef 282
dwmbe2be 282
dwmbe2le 283
dwmbe2lef 283
dwmle2be 284

- E -

E2BIG 57
EACCES 57
EADDRINUSE 57
EADDRNOTAVAIL 57
EADV 57
EAFNOSUPPORT 57
EAGAIN 57
EALREADY 57
EBADF 57
EBADFD 57
EBADMSG 57
EBADR 57
EBADRQC 57
EBADSLT 57
EBUSY 57
ecat_dev_on_off 284
ecat_get_state 287
ecat_probe 288
ecat_set_state 290
ECHILD 57
ECHRNG 57
ECOMM 57
ECONNABORTED 57
ECONNREFUSED 57
ECONNRESET 57
EDEADLK 57
EDESTADDRREQ 57
EDOM 57
EDOTDOT 57
EEXIST 57
EFAULT 57
EFBIG 57
EHOSTDOWN 57
EHOSTUNREACH 57
EIDRM 57
EILSEQ 57
EINPROGRESS 57
EINTR 57
EINVAL 57
EIO 57
EISCONN 57
EISDIR 57
EISNAM 57
ELIBACC 57
ELIBBAD 57
ELIBEXEC 57
ELIBMAX 57
ELIBSCN 57
ELNRNG 57
ELOOP 57
else 677
EMEDIUMTYPE 57
EMFILE 57
EMGSIZE 57
EMLINK 57
EMSGSIZE 57
EMULTIHOP 57
ENAMETOOLONG 57
end_aux 678
end_do_while 679
end_for 679
end_fun 680

© 2025 Robox SpA

end_main_loop 680
end_motion 681
end_ref 681
end_rule 682
end_select 682
end_while 683
Endianess conversion handling 103
Endianess conversion management 103
endif 683
ENETDOWN 57
ENETRESET 57
ENETUNREACH 57
ENFILE 57
ENOBUFS 57
ENODATA 57
ENODEV 57
ENOENT 57
ENOEXEC 57
ENOLCK 57
ENOLINK 57
ENOMEDIUM 57
ENOMEM 57
ENOMSG 57
ENONET 57
ENOPKG 57
ENOPROTOOPT 57
ENOSPC 57
ENOSR 57
ENOSTR 57
ENOSYS 57
ENOTBLK 57
ENOTCONN 57
ENOTDIR 57
ENOTEMPTY 57
ENOTSOCK 57
ENOTTY 57
ENOTUNIQ 57
ENXIO 57
EOPNOTSUPP 57
EOVERFLOW 57
EPERM 57
EPFNOSUPPORT 57
EPIPE 57
EPROTO 57
EPROTONOSUPPORT 57
EPROTTYPE 57
ERANGE 57
EREMCHG 57
EREMOTE 57
EREMOTEIO 57
ERESTART 57
EROFS 57
ESHUTDOWN 57
ESOCKTNOSUPPORT 57
ESPIPE 57
ESRCH 57
ESRMNT 57
ESTALE 57
ESTRPIPE 57
ETIME 57
ETIMEDOUT 57
ETOOMANYREFS 57
etsCHAR 891
ETXTBSY 57
EUCLEAN 57
EUNATCH 57
EUSERS 57
excl_lam 292, 684
excl_pam 293, 685
EXDEV 57

- F -

f_avail 293
f_close 294
f_copy 295
f_del 296
f_dir 297
f_format 298
f_gets 300
f_mkdir 301
f_open 302
f_puts 304
f_read 305
f_read_table 306
f_read_table_s 308
f_rename 310
f_seek 311
f_write 312
fall 314
fall_filt 314
fb_ctrlw 246
fb_stsw 246

© 2025 Robox SpA

fill_mem 685
filter_lp 317
filter_lp_st 317
filter_notch 319
filter_notch_st 320
first_time 321
Flash handling 115
Flash management 115
FLOAT 892
Flow control handling 102
Flow control management 102
for 686
force 322
fract 324
function 688
Functions error codes 57
Functions/instructions 19
Functions/instructions index 20

- G -

General index 17
get_float 325
get_hw_info 326
get_i16 328
get_i32 328
get_i64 692
get_i8 329
get_real 330
get_u16 331
get_u32 332
get_u64 693
get_u8 333
get_virt_ow 334
Global variable handling 127
Global variable management 127
goto 693
Grammar generalities 761
group 694
gv_create 334
gv_datatype 336
gv_datatype_byname 337
gv_destroy 338
gv_exist 338
gv_flags 339
gv_flags_byname 341
gv_link 342
gv_lock 344
gv_name 344
gv_read 345
gv_read_partial 345
gv_size 346
gv_size_byname 347
gv_unlink 347
gv_unlock 348
gv_update_file 348
gv_write 349
gv_write_partial 349

- I -

I/O forcing handling 116
I/O forcing management 116
I16 893
I32 894
I8 895
if 696
in_range 350
inc 351
index_of 352
inf_report 697
int_after 698
int_inp 699
int_timer 701
ipaddr_to_str 353
is_alarm_par_present 353
is_alarm_par_present_us 356
is_alarm_present 357
is_forced 358

- K -

K_1_PI 17
K_2PI 17
K_DUEPI 17
K_E 17
K_EPSILON 17
K_MAXFLOAT 17
K_MAXREAL 17
K_MINFLOAT 17
K_MINREAL 17
K_PI 17
K_PI_2 17
K_PIMEZZI 17
kin_conv 361

© 2025 Robox SpA

- L -

Linear axis: zero cycle towards the end of stroke 72
Linear axis: zero cycle towards the end of stroke + index 74
Linear axis: zero cycle towards the micro 66
Linear axis: zero cycle towards the micro + index 69
lit 702
LIVE Changes 903
log 362
log10 363
Logical operators 782
loop_time 363

- M -

M_ACCEL 61
M_ACCEL_LIMITED 61
M_ADJUSTING_DOWN 61
M_ADJUSTING_UP 61
M_AT_SPEED 61
M_DECEL 61
M_FOLLOWING 61
M_HOLDED 61
M_HOMING 61
M_ILLEGAL_ARGS 61
M_IN_ALARM 61
M_INITIALIZING 61
M_LAST_CYCLE 61
M_LOW_ACCELERATION 61
M_OUT_OF_LIMITS 61
M_OVERSHOOTING 61
M_POSITIONING 61
M_REACHED 61
M_SLAVE_EARLY_STOPPED 61
M_SLAVE_IS_EARLY 61
M_SLAVE_TOO_LATE 61
M_SPEED_LIMITED 61
M_SYNCHRONIZED 61
M_WAITING 61
M_WAITING_NEXT_CYCLE 61
master_filter 364
Mathematical functions/instructions 88
maximum 364
mbc_connect 365
mbc_init 366
mbc_mask_write_reg 368
mbc_mask_write_reg_s 370
mbc_read_coils 372
mbc_read_coils_s 373
mbc_read_discr_inp 374
mbc_read_discr_inp_s 376
mbc_read_hold_reg 377
mbc_read_hold_reg_s 379
mbc_read_inp_reg 381
mbc_read_inp_reg_s 382
mbc_shutdown 384
mbc_write_coils 384
mbc_write_coils_s 386
mbc_write_multiple_coils 387
mbc_write_multiple_coils_s 388
mbc_write_multiple_regs 390
mbc_write_multiple_regs_s 392
mbc_write_reg 393
mbc_write_reg_s 395
mbcu_mask_write_reg 396
mbcu_read_coils 398
mbcu_read_discr_inp 400
mbcu_read_hold_reg 401
mbcu_read_inp_reg 403
mbcu_write_coils 405
mbcu_write_multiple_coils 406
mbcu_write_multiple_regs 408
mbcu_write_reg 410
Metacommands 770
minimum 412
Miscellaneous 129
mod 412
Modbus error codes 62
Modbus/TCP client handling 116
Modbus/TCP client management 116
MOT handling 120
MOT management 120
mot_append 413
mot_clear 417
mot_create 417
mot_delete 421
mot_get 421
mot_remove 423
motion 703
Motion handling 120
Motion management 120

© 2025 Robox SpA

move_byte 704
move_mem 705
mt_ats 423
mt_ats2 424
mt_dis 706
mt_en 706
mt_rr_flag 707
mt_wait 707
mv_cam 425
mv_crimper 427
mv_follow 428
mv_follow2 432
mv_mot_exec 445
mv_phase_adj 447
mv_phase_adj2 448
mv_ramp 451
mv_reach_target 451
mv_sine_cam 453
mv_synchro 454
mv_table 455
mv_to 457, 459
mv_to (inline) 459
mv_to (struct) 457
mv_to_cj 461
mv_to_cjv 461
mv_to_cjv_info 463
mv_to_vel 464
mv_tracking 466
mva_follow2 442
mva_follow2_m 443
mva_jog 466
mva_jog2 468
mva_open_loop 470
mva_to_n 471
mva_to_n_cj 475
mva_to_n_v 479
mva_zc 486

- N -

normalize 488
not 782
Numerical relational operators 784

- O -

One-way rotary axis: zero cycle towards the micro + index with preliminary search for the micro variation 76
One-way rotary axis: zero cycle towards the micro + index without preliminary search for the micro variation 80
One-way rotary axis: zero cycle towards the micro with preliminary search for the micro variation 83
One-way rotary axis: zero cycle towards the micro without preliminary search for the micro variation 85
onoff_filt 489
Operators 774
or 782
order 708
out_m 710

- P -

par_even 491
par_odd 492
planar_comp_cp 493
planar_comp_ip 494
pos_capture_cmd 495
pos_capture_sts 499
pow 500
Power set handling 123
Power set management 123
POWER_SET 896
ps_channel_enable 501
ps_channel_enabled 504
ps_channel_fault 505
ps_channel_reset_fault 507
ps_drive_reset 509
ps_exclude_lax 511
ps_exclude_pax 513
ps_power_enable 514
ps_reinclude_lax 515
ps_reinclude_pax 516
ps_status 517

- Q -

qwbe2le 711
qwbe2lef 521
qwbedwx2le 711
qwbedwx2lef 521
qwle2be 712
qwle2bedwx 712
qwle2bedwxf 713
qwle2bef 714
qwle2mbe 714
qwle2mbedwx 715

© 2025 Robox SpA

qwle2mbedwxf 715
qwle2mbef 716
qwmbe2le 716
qwmbe2lef 522
qwmbedwx2le 717
qwmbedwx2lef 522

- R -

r_and 777
r_not 777
r_or 777
r_rol 523
r_ror 524
r_shl 525
r_shr 526
r_xor 777
r2m16 527
r2m32 528
r2m8 529
R3 language generalities 763
ramp 530
rand32 531
random 531
random_value 531
range 532
range_adj 533
rbit 534
REAL 898
ref 717
ref_prologue 718
release 534
restart_point 719
return 719
Return codes of the MV_ and MVA_ functions 61
rise 536
rise_filt 537
round 539
rtc_date 540
rtc_time 540
rule 720
Rule/task functions/instructions 91
rule_epilogue 728
rule_freq 729
rule_init 730
rule_periodic 730
rule_prologue 732
rule_start_norc 732

- S -

safe_appl_ack 541
safe_appl_diag 542
safe_diag 544
safe_diag_reset 546
safe_enab_out 547
safe_oper_ack 548
safe_prj_info 549
safe_sts 550
Safety functions/instructions 90
sb<n> 551
sb0 551, 733
sb1 552, 733
sb2 553, 733
sb3 553, 734
sb4 554, 734
sb5 554, 734
sb6 555, 735
sb7 556, 735
sbit 556
SDO Error Codes (CoE/SoE) 63
sdw<n> 557
sdw0 557, 735
sdw1 558, 736
select 736
ser_485_tx_sts 559
ser_bcc_h_init 561
ser_bin_in 562
ser_bin_out 563
ser_bin_ready 563
ser_buff_free 564
ser_buff_queue 564
ser_con 737
ser_flush 564
ser_in 565
ser_mod 566
ser_out 567
ser_r_sts 567
ser_ready 568
ser_w_sts 738
Serial channel handling 95
Serial channel management 95
set_float 739
set_i16 740

© 2025 Robox SpA

set_i32 741
set_i64 742
set_i8 743
set_kinematics 569
set_real 744
set_u16 745
set_u32 746
set_u64 747
set_u8 747
set_version 748
set_virt_iw 569
shift_left 570
shift_right 571
shl 573
shr 573
sign 574
similar 575
sin 576
sizeof 576
soe_ridn 577
soe_widn 589
sqrt 594
statistics 595
store_nvr_nopf 595
store_nvrr_nopf 596
str_asc 598
str_chr 598
str_compact 599
str_equal 600
str_find 600
str_format 601
str_i 602
str_insert 603
str_len 604
str_mid 605
str_r 606
str_replace 607
str_to_ipaddr 608
str_val 608
STRING 899
String concatenation operators 781
String handling 124
String management 124
STRU_AAX 795
STRU_AH_D 796
STRU_AH_D_EXT 797
STRU_AH_INFO 798
STRU_AS_D 799
STRU_AS_D_EXT 800
STRU_AXIO_PDI 801
STRU_BCC_TCP_CLIENT 803
STRU_CAM 804
STRU_CO_OBJ 805
STRU_COE_OBJ 807
STRU_COERRMSG 809
STRU_CRIMPER 812
STRU_ECAT_PROBE 814
STRU_FILEHND 815
STRU_FILEINFO 816
STRU_FOLLOW 816
STRU_FOLLOW2 819
STRU_FOLLOW2_INFO 827
STRU_FOLLOW2_S 827
STRU_FOLLOW2_SM 834
STRU_GROR 841
STRU_HW_INFO 841
STRU_INT 842
STRU_LARGE_INT 844
STRU_LARGE_UINT 844
STRU_MASTERF 845
STRU_MATCAM 846
STRU_MODBUS 847
STRU_MVGENITP 849
STRU_MVITP 850
STRU_MVITPCJ 852
STRU_MVTO 853
STRU_MVTOCJ 855
STRU_MVTOCJV 857
STRU_MVTOCJV_INFO 859
STRU_MVTOCJVE 861
STRU_MVZC 863
STRU_PHADJ 868
STRU_PHADJ2 871
STRU_PM_PD 875
STRU_REACHTARGET 876
STRU_SAFETY_PRJINFO 878
STRU_SOE_IDN 879
STRU_STAT 882
STRU_SYNCHRO 884
STRU_TRACK 886
STRUCT 888
STRUCT_P 889

© 2025 Robox SpA

sum16_mem 611
sum32_mem 612
sum8_mem 611
sumr_mem 612
sw<n> 612
sw0 612, 749
sw1 613, 749
sw2 614, 750
sw3 614, 750

- T -

TABSTR 889
tan 615
terminator 750
time_micro 616
TIMER 900
Timer handling 125
Timer management 125
tm_et 616, 751
tm_int 751
tm_of 753
tm_on 754
tm_pt 617, 756
tm_pt2 618, 756
tm_q 619
tm_reset 756
tm_sqw 756
tm_tp 758
tobcd 619
tobin 620
todeg 621
tolower 621
torad 621
toupper 622
tracking 622
trunc 624

- U -

U16 900
U32 901
U8 902
UDP/IP handling 126
UDP/IP management 126
udp_close 625
udp_open_client 626
udp_open_server 628
udp_recv 631
udp_recv_from 632
udp_recv_notify 634
udp_send 636
udp_send_notify 638
udp_send_to 640

- V -

virt_io 642

- W -

w<n> 644
w0 644, 759
w1 645, 759
w2 645, 760
w3 646, 760
wbe2le 647
wbit 647
while 760
wle2be 648
write_dis 648

- X -

xor 782

- Z -

Zero-cycles types 66

© 2025 Robox SpA