KUKA System Software

KUKA System Software 8.5

Operating and Programming Instructions for System Integrators

Issued: 19.03.2019
KSS 8.5 SI V7
KUKA Deutschland GmbH

© Copyright 2019

KUKA Deutschland GmbH
Zugspitzstraße 140
D-86165 Augsburg
Germany

This documentation or excerpts therefrom may not be reproduced or disclosed to third parties without the express permission of KUKA Deutschland GmbH.

Other functions not described in this documentation may be operable in the controller. The user has no claims to these functions, however, in the case of a replacement or service work.

We have checked the content of this documentation for conformity with the hardware and software described. Nevertheless, discrepancies cannot be precluded, for which reason we are not able to guarantee total conformity. The information in this documentation is checked on a regular basis, however, and necessary corrections will be incorporated in the subsequent edition.

Subject to technical alterations without an effect on the function.

KIM-PS5-DOC

Translation of the original documentation

Publication:	Pub KSS 8.5 SI (PDF) en
	PB6917
Book structure:	KSS 8.5 SI V9.1
	BS6445
Version:	KSS 8.5 SI V7

Contents

1 Introduction ... 17

1.1 Target group ... 17
1.2 Industrial robot documentation ... 17
1.3 Representation of warnings and notes ... 17
1.4 Trademarks ... 18
1.5 Licenses ... 18

2 Product description ... 19

2.1 Overview of the industrial robot ... 19
2.2 Overview of KUKA System Software (KSS) ... 19
2.3 System requirements ... 20
2.4 Intended use of the KUKA System Software ... 20
2.5 KUKA USB sticks ... 20

3 Safety ... 23

3.1 General ... 23
3.1.1 Liability ... 23
3.1.2 Intended use of the industrial robot ... 23
3.1.3 EC declaration of conformity and declaration of incorporation ... 24
3.1.4 Terms used ... 24
3.2 Personnel ... 27
3.3 Workspace, safety zone and danger zone ... 28
3.3.1 Determining stopping distances ... 28
3.4 Triggers for stop reactions ... 28
3.5 Safety functions ... 29
3.5.1 Overview of the safety functions ... 29
3.5.2 Safety controller ... 30
3.5.3 Selecting the operating mode ... 30
3.5.4 “Operator safety” signal ... 31
3.5.5 EMERGENCY STOP device ... 32
3.5.6 Logging off from the higher-level safety controller ... 32
3.5.7 External EMERGENCY STOP device ... 33
3.5.8 Enabling device ... 33
3.5.9 External enabling device ... 34
3.5.10 External safe operational stop ... 35
3.5.11 External safety stop 1 and external safety stop 2 ... 35
3.5.12 Velocity monitoring in T1 ... 35
3.6 Additional protective equipment ... 36
3.6.1 Jog mode ... 36
3.6.2 Software limit switches ... 36
3.6.3 Mechanical end stops ... 36
3.6.4 Mechanical axis limitation (optional) ... 36
3.6.5 Options for moving the manipulator without drive energy ... 37
3.6.6 Labeling on the industrial robot ... 37
3.6.7 External safeguards ... 37
3.7 Overview of operating modes and safety functions ... 38
3.8 Safety measures ... 38

3.8.1 General safety measures ... 38
3.8.2 IT security ... 40
3.8.3 Transportation ... 40
3.8.4 Start-up and recommissioning ... 41
3.8.4.1 Checking machine data and safety configuration ... 42
3.8.4.2 Start-up mode ... 44
3.8.5 Manual mode ... 45
3.8.6 Simulation ... 46
3.8.7 Automatic mode ... 47
3.8.8 Maintenance and repair ... 47
3.8.9 Decommissioning, storage and disposal ... 48
3.8.10 Safety measures for “single point of control” ... 48
3.9 Applied norms and regulations ... 50

4 Operation ... 53

4.1 KUKA smartPAD teach pendant ... 53
4.1.1 smartPAD ... 53
4.1.1.1 Front of smartPAD ... 53
4.1.1.2 Rear of smartPAD ... 55
4.1.2 smartPAD-2 ... 56
4.1.2.1 Front of smartPAD-2 ... 56
4.1.2.2 Rear of smartPAD-2 ... 58
4.1.3 Disconnecting and connecting the smartPAD ... 59
4.2 KUKA smartHMI user interface ... 61
4.2.1 Keypad ... 62
4.2.2 Status bar ... 63
4.2.3 Drives status indicator and Motion conditions window ... 64
4.2.4 Minimizing KUKA smartHMI (displaying Windows interface) ... 66
4.3 Switching on the robot controller and starting the KSS ... 66
4.4 Calling the main menu ... 66
4.5 Defining the start type for KSS ... 67
4.6 Shutting down or rebooting the robot controller ... 68
4.7 Switching drives on/off ... 71
4.8 Switching the robot controller off ... 71
4.9 Setting the user interface language ... 72
4.10 Creating a screenshot on the smartPAD ... 72
4.11 Online documentation and help for messages ... 72
4.11.1 Calling online documentation ... 72
4.11.2 Calling help for the messages ... 73
4.12 Changing user group ... 76
4.13 User groups ... 77
4.14 Changing operating mode ... 78
4.15 Coordinate systems ... 79
4.16 Jogging the robot ... 80
4.16.1 “Jog options” window ... 81
4.16.1.1 General tab ... 81
4.16.1.2 Keys tab ... 82
4.16.1.3 Mouse tab ... 83
4.16.1.4 KCP pos. tab ... 84

4.16.1.5 Cur. tool/base tab ... 84
4.16.1.6 Collision detection tab ... 85
4.16.2 Setting the jog override ... 86
4.16.3 Selecting the tool and base ... 87
4.16.4 Axis-specific jogging with the jog keys ... 87
4.16.5 Cartesian jogging with the jog keys ... 88
4.16.6 Configuring the Space Mouse ... 88
4.16.7 Defining the alignment of the Space Mouse ... 90
4.16.8 Cartesian jogging with the Space Mouse ... 91
4.16.9 Incremental jogging ... 91
4.16.10 Backward motion using the jog keys ... 92
4.16.10.1 Backward motion using the jog keys – Overview ... 92
4.16.10.2 Recording in buffer ... 93
4.16.10.3 Executing motions backwards (using jog keys) ... 95
4.17 Jogging external axes ... 95
4.18 Bypassing workspace monitoring ... 96
4.19 Display functions ... 97
4.19.1 Displaying the actual position ... 97
4.19.1.1 Window Actual position, view Cartesian ... 97
4.19.1.2 Window Actual position, view Axis-specific ... 98
4.19.2 Displaying digital inputs/outputs ... 100
4.19.3 Displaying analog inputs/outputs ... 101
4.19.4 Displaying inputs/outputs for Automatic External ... 102
4.19.5 Displaying and modifying the value of a variable ... 103
4.19.6 Displaying the state of a variable ... 105
4.19.7 Displaying the variable overview and modifying variables ... 105
4.19.8 Displaying cyclical flags ... 107
4.19.9 Displaying flags ... 108
4.19.10 Displaying counters ... 109
4.19.11 Displaying timers ... 110
4.19.12 Displaying information about the robot and robot controller ... 111
4.19.13 Displaying/editing robot data ... 112
4.19.14 Measuring and displaying energy consumption ... 114
4.19.14.1 $ENERGY_MEASURING: starting and stopping consumption measurement ... 116
4.19.14.2 $ENERGYMODULE: energy model available for robot? ... 117
4.19.14.3 $ENERGY_CONFIG_STATE: status of the energy model for each axis ... 118
4.19.14.4 $ENERGY_PERIOD: energy consumption of the last 60 min ... 119
4.19.14.5 $ENERGY_TOTAL: energy consumption since last cold start ... 120
4.19.14.6 $ENERGY_INTERIM: energy consumption between time stamps ... 120
4.19.14.7 INTERIMENERGY(): Setting a time stamp for the measurement of energy consumption ... 121
4.20 Displaying the battery state ... 122

5 Start-up and recommissioning ... 125

5.1 Protecting the Windows system ... 125
5.2 Switching on the robot controller with the kinematic system for the first time ... 125
5.3 Changing the password for a Windows system ... 129
5.3.1 Changing the password: return values ... 130
5.4 Start-up wizard ... 132

5.5 Modifying the machine data configuration 132
5.6 Defining hardware options 133
5.7 Preventing an unexpected start of the peripheral devices 135
5.7.1 Behavior with/without additional enabling 135
5.7.2 Configuring additional enabling 135
5.7.3 SIGNAL $CRIT_PERI_ACK 136
5.7.4 SIGNAL $CRIT_PERI_ACK_REQ 137
5.8 Changing the safety ID of the PROFINET device 139
5.9 Switching safe Cartesian monitoring on/off 140
5.10 Jogging the robot without a higher-level safety controller 141
5.11 Checking the activation of the positionally accurate robot model 142
5.12 Activating palletizing mode 143
5.13 Mastering 144
5.13.1 Mastering methods 145
5.13.2 Moving axes to the pre-mastering position using mastering marks 147
5.13.3 Moving axes to the pre-mastering position using the probe 148
5.13.4 Mastering LEDs 150
5.13.5 Mastering with the SEMD 151
5.13.5.1 First mastering (with SEMD) 151
5.13.5.2 Teach offset (with SEMD) 154
5.13.5.3 Checking load mastering with offset (with SEMD) 155
5.13.6 Mastering with the dial gauge 157
5.13.7 Mastering external axes 158
5.13.8 Reference mastering 159
5.13.9 Mastering with the MEMD and mark 160
5.13.9.1 Moving A6 to the mastering position (with line mark) 160
5.13.9.2 First mastering (with MEMD) 161
5.13.9.3 Teach offset (with MEMD) 164
5.13.9.4 Checking load mastering with offset (with MEMD) 165
5.13.10 Manually unmastering axes 166
5.14 Modifying software limit switches 167
5.15 Calibration 169
5.15.1 Defining the tool direction 169
5.15.2 Introduction to TOOL calibration 170
5.15.3 Calibrate TOOL or enter it numerically (tool/workpiece on flange) 171
5.15.4 Introduction to BASE calibration 172
5.15.5 Calibrate BASE or enter it numerically (base/fixed tool) 172
5.15.6 “Tool/base management” window 173
5.15.6.1 Tool/base management window – “Overview” area 173
5.15.6.2 Icons in the “Overview” area 175
5.15.6.3 Tool/base management window – “Editing” area 176
5.15.6.4 Tool/base management window – “Calibration” area 177
5.15.7 Overview of calibration methods 178
5.15.7.1 XYZ 4-point method 179
5.15.7.2 XYZ Reference method 180
5.15.7.3 XYZ method 180
5.15.7.4 ABC world method 181
5.15.7.5 ABC 2-point method 182
5.15.7.6 3-point method 183

5.15.7.7 Indirect method ... 185
5.15.8 Linear unit ... 186
5.15.8.1 Checking whether the linear unit needs to be calibrated ... 187
5.15.8.2 Calibrating the linear unit ... 188
5.15.8.3 Entering the linear unit numerically ... 189
5.16 Load data ... 190
5.16.1 Load data parameters ... 190
5.16.2 Entering payload data numerically ... 191
5.16.3 Entering supplementary load data numerically ... 192
5.16.4 Load data verification ... 192
5.17 Exporting/importing long texts ... 194
5.18 Adapting the MAMES values after exchanging the in-line wrist ... 196
5.19 Maintenance handbook ... 198
5.19.1 Logging maintenance ... 198
5.19.2 Displaying a maintenance log ... 200

6 Configuration ... 201

6.1 Configuring the KUKA Line Interface (KLI) ... 201
6.1.1 Configuring the Windows interface (without field bus) ... 201
6.1.2 Configuring the field bus interface and creating the Windows interface ... 203
6.1.3 Displaying ports of the Windows interface or enabling an additional port ... 205
6.1.4 Displaying or modifying filters ... 206
6.1.5 Error display in the Address and Subnet boxes ... 206
6.2 Displaying the subnet configuration of the robot controller ... 207
6.3 Reconfiguring the I/O driver ... 208
6.4 Configuring safe axis monitoring functions ... 209
6.4.1 Parameter: Braking time ... 210
6.4.2 Checking the limits for the maximum axis velocity in T1 mode ... 213
6.5 Checking the values for the safe axis monitoring functions ... 214
6.6 Checking the safety configuration of the robot controller ... 215
6.7 Checksum of the safety configuration ... 216
6.8 Exporting the safety configuration (XML export) ... 217
6.9 Configuring the variable overview ... 217
6.10 Changing the password ... 219
6.11 Displaying and modifying user rights ... 219
6.11.1 Information about function groups in this documentation ... 220
6.11.2 Overview of function groups ... 221
6.12 Enabling or disabling operating modes for specific user groups ... 223
6.13 Energy saving mode ($ECO_LEVEL) ... 224
6.14 Configuring workspaces ... 225
6.14.1 Configuring Cartesian workspaces ... 226
6.14.2 Modes for Cartesian workspaces ... 228
6.14.3 Configuring axis-specific workspaces ... 229
6.14.4 Modes for axis-specific workspaces ... 231
6.15 Defining limits for reteaching ... 231
6.16 Warm-up ... 233
6.16.1 Warm-up sequence ... 233
6.16.2 Configuring warm-up ... 234
6.17 Defining calibration tolerances ... 236

6.18 Configuring backward motion ... 237
6.19 Configuring Automatic External ... 238
6.19.1 Configuring CELL.SRC ... 239
6.19.2 Configuring Automatic External inputs/outputs ... 240
6.19.2.1 Automatic External inputs ... 241
6.19.2.2 Odd / even parity ... 244
6.19.2.3 Automatic External outputs ... 245
6.19.3 Transmitting error numbers to the higher-level controller ... 247
6.19.4 Signal diagrams ... 249
6.20 Torque mode ... 255
6.20.1 Overview of torque mode ... 255
6.20.1.1 Using torque mode ... 255
6.20.1.2 Robot program example: setting A1 to “soft” in both directions ... 257
6.20.2 Activating torque mode: SET_TORQUE_LIMITS() ... 258
6.20.3 Deactivating torque mode: RESET_TORQUE_LIMITS() ... 261
6.20.4 Interpreter specifics ... 262
6.20.5 Diagnostic variables for torque mode ... 263
6.20.5.1 $TORQUE_AXIS_ACT ... 263
6.20.5.2 $TORQUE_AXIS_MAX_0 ... 264
6.20.5.3 $TORQUE_AXIS_MAX ... 264
6.20.5.4 $TORQUE_AXIS_LIMITS ... 264
6.20.5.5 $HOLDING_TORQUE ... 265
6.20.5.6 Comparison: $TORQUE_AXIS_ACT and $HOLDING_TORQUE ... 266
6.20.6 Other examples ... 266
6.20.6.1 Robot program: setting axis to “soft” in both directions ... 266
6.20.6.2 Robot program: avoiding damage in the event of collisions ... 267
6.20.6.3 Robot program: torque mode in the interrupt ... 268
6.20.6.4 Robot program: servo gun builds up pressure ... 269
6.20.6.5 Submit program: servo gun builds up pressure ... 271
6.21 Event planner: configuring a data comparison ... 272
6.22 Brake test ... 272
6.22.1 Overview of the brake test ... 272
6.22.2 Sequence when testing a brake ... 274
6.22.3 Programs for the brake test ... 274
6.22.4 Overview of the brake test setup ... 275
6.22.4.1 Activating the brake test, defining the cycle time and axes ... 275
6.22.4.2 “Brake test configuration” window ... 276
6.22.4.3 Configuring input and output signals for the brake test ... 277
6.22.4.4 Signal diagram of the brake test – examples ... 279
6.22.4.5 Teaching positions for the brake test ... 281
6.22.4.6 Testing the sequence in the case of defective brakes ... 282
6.22.5 Performing a brake test ... 283
6.22.5.1 Performing a brake test for requested axes (cyclically via program) ... 283
6.22.5.2 Performing a brake test for active axes (manually) ... 285
6.22.5.3 Performing a brake test for further axes (e.g. couplable axes) ... 286
6.22.6 Automatic brake check ... 287
6.22.7 System functions for the brake test ... 288
6.22.7.1 GET_AXESMASK() ... 288
6.22.7.2 GET_BRAKETEST_TIME() ... 289

7 Program and project management ... 291

7.1 Creating a new program ... 291
7.2 Creating a new folder ... 291
7.3 Renaming a file or folder ... 291
7.4 Navigator file manager ... 292
7.4.1 Selecting filters ... 293
7.4.2 Displaying or modifying properties of files and folders ... 293
7.5 Selecting or opening a program ... 297
7.5.1 Selecting and deselecting a program ... 297
7.5.2 Opening a program ... 298
7.5.3 Toggling between the Navigator and the program ... 299
7.6 Structure of a KRL program ... 300
7.6.1 HOME position ... 300
7.7 Displaying/hiding program sections ... 301
7.7.1 Displaying/hiding the DEF line ... 301
7.7.2 Activating detail view ... 301
7.7.3 Activating/deactivating the line break function ... 302
7.7.4 Displaying folds ... 302
7.8 Editing programs ... 304
7.8.1 Inserting a comment or stamp ... 304
7.8.2 Selecting a range ... 305
7.8.3 Deleting program lines ... 305
7.8.4 Creating folds ... 305
7.8.5 Additional editing functions ... 306
7.9 Printing a program ... 307
7.10 Archiving and restoring data ... 307
7.10.1 Archiving overview ... 307
7.10.2 Archiving to a USB stick ... 308
7.10.3 Archiving on the network ... 309
7.10.4 Archiving the logbook ... 310
7.10.5 Restoring data ... 310
7.11 Project management ... 311
7.11.1 Pinning a project on the robot controller ... 311
7.11.2 Activating an existing project ... 311
7.11.3 Project management window ... 313
7.11.3.1 Projects tab ... 313
7.11.3.2 Restoration points tab ... 315
7.12 Backup Manager ... 316
7.12.1 Overview of Backup Manager ... 316
7.12.2 Backing up projects, option packages and RDC data manually ... 317
7.12.3 Manually restoring projects and option packages ... 318
7.12.4 Restoring RDC data manually ... 320
7.12.5 Configuring Backup Manager ... 321
7.12.5.1 “Backup configuration” tab ... 322
7.12.5.2 “Signal interface” tab ... 324

8 Program execution ... 327

8.1 Selecting the program run mode ... 327

8.2 Program run modes ... 327
8.3 Advance run ... 328
8.4 Block pointer ... 329
8.5 Setting the program override ... 331
8.6 Robot interpreter status indicator ... 332
8.7 Starting a program forwards (manual) ... 332
8.8 Starting a program forwards (automatic) ... 333
8.9 BCO run ... 333
8.10 Carrying out a block selection ... 334
8.11 Resetting a program ... 335
8.12 Starting Automatic External mode ... 335
8.13 Backward motion using the Start backwards key ... 336
8.13.1 Executing motions backwards (using the “Start backwards” key) ... 336
8.13.2 Functional principle and characteristics of backward motion ... 337
8.13.2.1 Response in the case of subprograms ... 338
8.13.2.2 Approximate positioning response ... 338
8.13.2.3 Response in the case of weave motions ... 339
8.13.2.4 Switching from backwards to forwards ... 340
8.13.3 System variables with changed meaning ... 341
8.13.4 Comparison of “Start backwards”/backwards using the jog keys ... 341
8.14 Collision detection ... 342
8.14.1 Overview of collision detection ... 342
8.14.2 Resuming motion after a collision ... 342
8.14.3 Configuring collision detection for jogging ... 343
8.14.4 “Collision detection - Jogging configuration” window ... 344
8.14.5 Configuring collision detection for program mode / filling data sets ... 345
8.14.5.1 Filling data sets via smartHMI ... 345
8.14.5.2 “Collision detection - Data set configuration” window ... 346
8.14.5.3 Filling data sets via the program with “SaveMax” ... 347
8.14.5.4 Inline form COLLDETECT SaveMax ... 348
8.14.6 Activating/deactivating collision detection via inline form ... 348
8.14.7 Displaying current values / Collision detection - Display window ... 349
8.14.8 Collision detection – System variables ... 350

9 Basic principles of motion programming ... 355

9.1 Overview of motion types ... 355
9.2 Motion type PTP ... 355
9.3 Motion type LIN ... 356
9.4 Motion type CIRC ... 356
9.5 Approximate positioning ... 357
9.6 Orientation control LIN, CIRC ... 358
9.6.1 Combinations of $ORI_TYPE and $CIRC_TYPE ... 359
9.7 Spline motion type ... 361
9.7.1 Velocity profile for spline motions ... 363
9.7.2 BCO run with spline motions via the Block selection button ... 364
9.7.3 BCO run with spline motions after program modification ... 366
9.7.3.1 BCO run following modification to the current spline block ... 366
9.7.3.2 BCO run following modification of another spline block ... 367
9.7.4 Modifications to spline blocks ... 369

9.7.5 Approximation of spline motions ... 372
9.7.6 Replacing an approximated CP motion with a spline block ... 372
9.7.6.1 SLIN-SPL-SLIN transition ... 375
9.8 Orientation control for CP spline motions ... 376
9.8.1 SCIRC: reference system for the orientation control ... 378
9.8.2 SCIRC: orientation behavior ... 379
9.8.2.1 SCIRC: Orientation behavior – example: auxiliary point ... 381
9.8.2.2 SCIRC: Orientation behavior – example: end point ... 382
9.9 Circular angle ... 383
9.10 Status and Turn ... 384
9.10.1 Status ... 385
9.10.2 Turn ... 388
9.11 Singularities ... 388

10 Programming with inline forms ... 391

10.1 Instructions for programming ... 391
10.2 Names in inline forms ... 391
10.3 Programming PTP, LIN and CIRC motions ... 391
10.3.1 Programming a PTP motion ... 391
10.3.2 Inline form “PTP” ... 392
10.3.3 Programming a LIN motion ... 393
10.3.4 Inline form “LIN” ... 393
10.3.5 Programming a CIRC motion ... 394
10.3.6 Inline form “CIRC” ... 394
10.3.7 Option window “Frames” ... 395
10.3.8 Option window “Motion parameters” (LIN, CIRC, PTP) ... 396
10.4 Programming spline motions ... 397
10.4.1 Programming tips for spline motions ... 397
10.4.2 Programming a spline block ... 398
10.4.2.1 Inline form for CP spline block ... 399
10.4.2.2 Inline form “PTP SPLINE block” ... 400
10.4.2.3 Option window “Motion parameters” (CP spline block) ... 401
10.4.2.4 Option window “Motion parameters” (PTP spline block) ... 402
10.4.3 Programming segments for a spline block ... 402
10.4.3.1 Programming an SPL or SLIN segment ... 402
10.4.3.2 Inline form SPL spline segment/SLIN spline segment ... 403
10.4.3.3 Programming an SCIRC segment ... 404
10.4.3.4 Inline form SCIRC spline segment ... 404
10.4.3.5 Programming an SPTP segment ... 405
10.4.3.6 Inline form for SPTP segment ... 406
10.4.3.7 Option window “Motion parameters” (CP spline segment) ... 407
10.4.3.8 Option window “Motion parameters” (SPTP) ... 408
10.4.3.9 Option window “Logic parameters” ... 409
10.4.3.10 Teaching the shift in space for logic parameters ... 412
10.4.4 Programming individual spline motions ... 413
10.4.4.1 Programming an individual SLIN motion ... 413
10.4.4.2 Inline form “SLIN” ... 413
10.4.4.3 Option window “Motion parameters” (SLIN) ... 414
10.4.4.4 Programming an individual SCIRC motion ... 415

10.4.4.5 Inline form “SCIRC”... 415
10.4.4.6 Option window “Motion parameters” (SCIRC)... 417
10.4.4.7 Programming an individual SPTP motion... 418
10.4.4.8 Inline form “SPTP”... 418
10.4.5 Conditional stop... 419
10.4.5.1 Inline form “Spline Stop Condition”... 420
10.4.5.2 Stop condition: example and braking characteristics... 421
10.4.6 Constant velocity range in the CP spline block... 423
10.4.6.1 Block selection to the constant velocity range... 423
10.4.6.2 Maximum limits... 423
10.5 Displaying the distance between points... 424
10.6 Modifying programmed motions... 425
10.6.1 Modifying motion parameters... 425
10.6.2 Modifying blocks of motion parameters... 425
10.6.3 Re-teaching a point... 425
10.6.4 Transforming blocks of coordinates... 426
10.6.4.1 “Axis mirroring” window... 429
10.6.4.2 “Transform - Axis Specific” window... 430
10.6.4.3 “Transform - Cartesian Base” window... 431
10.7 Global points... 432
10.7.1 Global points – overview... 432
10.7.2 Creating a global point (creating new point or converting local point)... 433
10.7.3 Modifying a global point... 434
10.8 Programming logic instructions... 435
10.8.1 Inputs/outputs... 435
10.8.2 Setting a digital output - OUT... 436
10.8.3 Inline form “OUT”... 436
10.8.4 Setting a pulse output - PULSE... 437
10.8.5 Inline form “PULSE”... 437
10.8.6 Setting an analog output - ANOUT... 437
10.8.7 Inline form “ANOUT” (static)... 438
10.8.8 Inline form “ANOUT” (dynamic)... 438
10.8.9 Programming a wait time - WAIT... 439
10.8.10 Inline form “WAIT”... 439
10.8.11 Programming a signal-dependent wait function - WAITFOR... 440
10.8.12 Inline form “WAITFOR”... 440
10.8.13 Switching on the path - SYN OUT... 441
10.8.14 Inline form “SYN OUT”, option “START/END”... 441
10.8.15 Inline form “SYN OUT”, option “PATH”... 444
10.8.16 Setting a pulse on the path - SYN PULSE... 446
10.8.17 Inline form “SYN PULSE”... 447
10.8.18 Modifying a logic instruction... 448

11 KRL programming... 449

11.1 Instructions for programming... 449
11.2 Symbols and fonts... 449
11.3 Important KRL terms... 450
11.3.1 SRC files and DAT files... 450
11.3.2 Naming conventions and keywords... 450

11.3.3 Data types ... 451
11.3.4 Areas of validity ... 453
11.3.4.1 Making subprograms, functions and interrupts available globally ... 453
11.3.4.2 Making variables, constants, signals and user data types available globally ... 454
11.3.5 Constants ... 455
11.4 Variables and declarations ... 455
11.4.1 DECL: declaring variables, arrays and constants ... 455
11.4.2 ENUM: defining an enumeration type ... 457
11.4.3 STRUC: defining a structure type ... 458
11.5 Motion programming: PTP, LIN, CIRC ... 459
11.5.1 PTP: motion programming ... 459
11.5.2 LIN: motion programming ... 460
11.5.3 CIRC: motion programming ... 461
11.5.4 PTP_REL: relative motion programming ... 462
11.5.5 LIN_REL: relative motion programming ... 463
11.5.6 CIRC_REL: relative motion programming ... 465
11.5.7 Approximation parameters for PTP, LIN CIRC and …_REL ... 466
11.5.8 REL motions for infinitely rotating axes ... 467
11.6 Motion programming: spline ... 469
11.6.1 SPLINE ... ENDSPLINE: programming a CP spline block ... 469
11.6.2 PTP_SPLINE ... ENDSPLINE: programming a PTP spline block ... 470
11.6.3 SLIN: motion programming ... 471
11.6.4 SCIRC: motion programming ... 472
11.6.5 SPL: motion programming ... 473
11.6.6 SPTP: motion programming ... 474
11.6.7 SLIN_REL: relative motion programming ... 475
11.6.8 SCIRC_REL: relative motion programming ... 477
11.6.9 SPL_REL: relative motion programming ... 478
11.6.10 SPTP_REL: relative motion programming ... 479
11.6.11 System variables for WITH ... 480
11.6.12 TIME_BLOCK: programming a time block for spline ... 481
11.6.13 CONST_VEL: programming a constant velocity range for spline motions ... 485
11.6.13.1 System variables for CONST_VEL ... 486
11.6.14 STOP WHEN PATH: programming a conditional stop for spline ... 487
11.6.15 $EX_AX_IGNORE ... 488
11.7 Program execution control ... 489
11.7.1 CONTINUE: preventing an advance run stop ... 489
11.7.2 EXIT: exiting a loop ... 490
11.7.3 FOR ... TO ... ENDFOR: programming a counting loop ... 490
11.7.4 GOTO: jump to position in the program ... 491
11.7.5 HALT: stopping a program ... 492
11.7.6 IF ... THEN ... ENDIF: programming a conditional branch ... 493
11.7.7 LOOP ... ENDLOOP: programming an endless loop ... 493
11.7.8 ON_ERROR_PROCEED: Skipping an instruction that causes an interpreter error ... 494
11.7.8.1 Handlable/non-handlable errors ... 495
11.7.8.2 ON_ERROR_PROCEED with motion instructions ... 496
11.7.8.3 Examples for ON_ERROR_PROCEED ... 497
11.7.8.4 $ERR ... 498

11.7.9 REPEAT ... UNTIL: programming a post-test loop ... 502
11.7.10 SWITCH ... CASE ... ENDSWITCH: programming a multiple branch ... 502
11.7.11 WAIT FOR … : waiting until a condition has been met ... 504
11.7.12 WAIT SEC … : programming a wait time ... 504
11.7.13 WHILE ... ENDWHILE: programming a rejecting loop ... 505
11.8 Inputs/outputs ... 506
11.8.1 ANIN: cyclically reading an analog input ... 506
11.8.2 ANOUT: writing cyclically to an analog output ... 507
11.8.3 PULSE: setting a pulse output ... 508
11.8.4 SIGNAL: signal declaration for inputs/outputs ... 512
11.9 Subprograms and functions ... 513
11.9.1 Calling a subprogram ... 513
11.9.2 Calling a function ... 514
11.9.3 DEFFCT ... ENDFCT: defining a function ... 514
11.9.4 RETURN: jump back to the calling program ... 515
11.9.5 Transferring parameters to a subprogram or function ... 516
11.9.6 Transferring a parameter to a different data type ... 520
11.10 Interrupt programming ... 520
11.10.1 INTERRUPT … DECL … WHEN … DO … : declaring an interrupt ... 520
11.10.2 INTERRUPT ON/OFF: activating or deactivating an interrupt ... 523
11.10.3 INTERRUPT DISABLE/ENABLE: disabling or enabling an interrupt ... 524
11.10.4 BRAKE: stopping the robot from an interrupt program ... 526
11.10.5 RESUME: cancelling interrupt programs ... 527
11.11 Path-related switching actions (=Trigger) ... 529
11.11.1 TRIGGER WHEN DISTANCE ... 529
11.11.2 TRIGGER WHEN PATH ... 533
11.11.2.1 Reference point for approximate positioning – overview ... 537
11.11.2.2 Reference point for homogenous approximate positioning ... 537
11.11.2.3 Reference point for mixed approximate positioning (spline) ... 539
11.11.2.4 Reference point for mixed approximate positioning (LIN/CIRC/PTP) ... 540
11.11.3 Constraints for functions in the trigger ... 540
11.11.4 Useful system variables for working with PATH triggers ... 541
11.11.4.1 $DIST_NEXT ... 541
11.11.4.2 $DIST_LAST ... 541
11.12 Communication ... 542
11.13 Operators ... 542
11.13.1 Arithmetic operators ... 542
11.13.2 Geometric operator ... 543
11.13.2.1 Sequence of the operands ... 544
11.13.2.2 Example of a double operation ... 545
11.13.3 Relational operators ... 546
11.13.4 Logic operators ... 547
11.13.5 Bit operators ... 548
11.13.6 Priority of the operators ... 550
11.14 Mathematical standard functions ... 551
11.15 System functions ... 553
11.15.1 DELETE_BACKWARD_BUFFER(): preventing backward start ... 553
11.15.2 FORWARD(): converting axis angles to a Cartesian position ... 553
11.15.3 INV_POS(): calculating an inverse position ... 554

11.15.4 INVERSE(): converting a Cartesian position to axis angles ... 556
11.15.5 ROB_STOP(): preventing robot motions ... 558
11.15.6 SET_BRAKE_DELAY(): reducing the brake delay ... 560
11.15.7 VARSTATE(): polling the state of a variable ... 563
11.16 Editing string variables ... 564
11.16.1 Converting a string variable to a different data type ... 564
11.16.2 String variable length in the declaration ... 566
11.16.3 String variable length after initialization ... 567
11.16.4 Deleting the contents of a string variable ... 567
11.16.5 Extending a string variable ... 568
11.16.6 Searching a string variable ... 569
11.16.7 Comparing the contents of string variables ... 569
11.16.8 Copying a string variable ... 570

12 Submit interpreter ... 573

12.1 Function of the submit interpreters ... 573
12.2 Status indicator for the submit interpreter ... 574
12.3 Operation of submit interpreter ... 575
12.3.1 Automatic starting of submit interpreters (on cold start) ... 575
12.3.2 Manual control of submit interpreters ... 576
12.3.2.1 Stopping all submit interpreters ... 576
12.3.2.2 Deselecting all submit interpreters ... 576
12.3.2.3 Starting all submit interpreters ... 577
12.3.2.4 Stopping, resetting or deselecting individual submit interpreters ... 577
12.3.2.5 Starting individual submit interpreters ... 578
12.3.2.6 Starting the system submit interpreter individually ... 578
12.3.3 “Submit interpreter” window ... 579
12.3.3.1 Current display/assignment tab ... 579
12.3.3.2 “Cold start configuration” tab ... 580
12.4 Submit interpreter programming ... 581
12.4.1 Creating a new SUB program ... 581
12.4.2 Editing the sps.sub program ... 582
12.4.3 KRL instructions not allowed or subject to restrictions ... 584
12.4.4 Subprograms ... 584
12.4.5 Inputs/outputs and communication ... 585
12.4.6 System variables ... 585
12.4.6.1 Changes in relation to Single Submit mode ... 586
12.4.6.2 $INTERPRETER ... 587
12.4.6.3 $PROG_INFO[] ... 587
12.4.7 CWRITE: changes in relation to Single Submit mode ... 588
12.5 Overview: changes of state due to actions in the menu ... 590

13 Diagnosis ... 593

13.1 Logbook ... 593
13.1.1 Displaying the logbook ... 593
13.1.2 “Log” tab ... 593
13.1.3 Filter tab ... 595
13.1.4 Configuring the logbook ... 595
13.2 Displaying the caller stack ... 596

13.3 Displaying interrupts ... 597
13.4 Displaying diagnostic data about the kernel system ... 599
13.5 Automatically compressing data for error analysis (KRCDiag) ... 599

14 Installation ... 601

14.1 System requirements ... 601
14.2 Installing additional software ... 601
14.3 KSS update ... 603
14.3.1 Update from USB stick ... 603
14.3.2 Update from the network ... 604
14.4 Automatically updating the firmware of hardware components ... 605

15 Appendix ... 607

15.1 Assignment of functions and function groups ... 607
15.1.1 Menu item File ... 607
15.1.2 Menu item Configuration ... 608
15.1.2.1 Automatic External ... 609
15.1.2.2 I/O drivers ... 610
15.1.2.3 Cartesian workspaces/Axis-specific workspaces ... 610
15.1.2.4 Event planner ... 610
15.1.2.5 Point coordinate correction limit ... 610
15.1.3 Menu item Display ... 610
15.1.3.1 Variable display ... 611
15.1.3.2 Variable overview ... 611
15.1.4 Menu item Diagnosis ... 611
15.1.4.1 Trace ... 612
15.1.4.2 TraceLogbook ... 612
15.1.5 Menu item Start-up ... 612
15.1.5.1 Robot data ... 613
15.1.5.2 Network configuration ... 614
15.1.5.3 Additional software ... 614
15.1.6 Menu item Shutdown ... 614
15.1.7 Menu item Help ... 615
15.1.8 Navigator ... 615
15.1.9 Navigator: Menu Edit ... 616
15.1.10 Editor: button bar ... 616
15.1.11 “Project management” window ... 617
15.1.12 Jog options window: Tabs ... 617
15.1.13 Wait messages ... 618
15.1.14 Editor: Menu Edit ... 618
15.1.15 Editor: Commands menu ... 619
15.1.16 smartHMI: status bar ... 620
15.1.17 smartHMI: left sidebar ... 621
15.1.18 smartHMI: right sidebar ... 621

16 KUKA Service ... 623

16.1 Requesting support ... 623
16.2 KUKA Customer Support ... 623

Index 631

1 Introduction

1.1 Target group

This documentation is aimed at users with the following knowledge and skills:

- Advanced knowledge of the robot controller system
- Advanced KRL programming skills

For optimal use of our products, we recommend that our customers take part in a course of training at KUKA College. Information about the training program can be found at www.kuka.com or can be obtained directly from our subsidiaries.

1.2 Industrial robot documentation

The industrial robot documentation consists of the following parts:

- Documentation for the robot arm
- Documentation for the robot controller
- Documentation for the smartPAD-2
- Operating and programming instructions for the System Software
- Instructions for options and accessories
- Spare parts in KUKA.Xpert

Each of these sets of instructions is a separate document.

1.3 Representation of warnings and notes

Safety

These warnings are provided for safety purposes and **must** be observed.

DANGER

These warnings mean that it is certain or highly probable that death or severe injuries **will** occur, if no precautions are taken.

WARNING

These warnings mean that death or severe injuries **may** occur, if no precautions are taken.

CAUTION

These warnings mean that minor injuries **may** occur, if no precautions are taken.

NOTICE

These warnings mean that damage to property **may** occur, if no precautions are taken.

These warnings contain references to safety-relevant information or general safety measures.
These warnings do not refer to individual hazards or individual precautionary measures.

This warning draws attention to procedures which serve to prevent or remedy emergencies or malfunctions:

SAFETY INSTRUCTION
The following procedure must be followed exactly!

Procedures marked with this warning **must** be followed exactly.

Notices

These notices serve to make your work easier or contain references to further information.

Tip to make your work easier or reference to further information.

1.4 Trademarks

Windows is a trademark of Microsoft Corporation.

WordPad is a trademark of Microsoft Corporation.

1.5 Licenses

The KUKA license conditions and the license conditions of the open-source software used can be found here:

- Under .\LICENSE on the data storage medium with the installation files of the KUKA software

Further information about open-source licenses can be requested from the following address: opensource@kuka.com

The end user license agreement can also be viewed at the following point in the System Software:

- KUKA smartHMI, main menu:
  Menu item **Help** > **Info**, tab **EULA**

2 Product description

2.1 Overview of the industrial robot

The industrial robot consists of the following components:

- Manipulator
- Robot controller
- Teach pendant
- Connecting cables
- Software
- Options, accessories

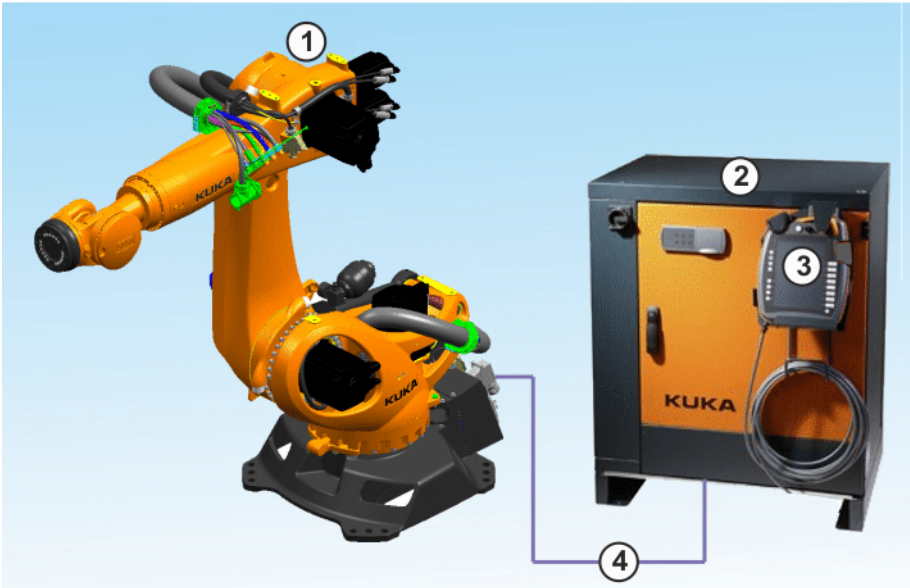


Fig. 2-1: Example of an industrial robot

1	Manipulator	3	KUKA smartPAD
2	Robot controller	4	Connecting cables

2.2 Overview of KUKA System Software (KSS)

Description

The KUKA System Software (KSS) is responsible for all the basic operator control functions of the industrial robot.

- Path planning
- I/O management
- Data and file management
- etc.

Additional technology packages, containing application-specific instructions and configurations, can be installed.

smartHMI

The user interface of the KUKA System Software is called KUKA smartHMI (smart Human-Machine Interface).

Features:

- User administration
- Program editor
- KRL (KUKA Robot Language)
- Inline forms for programming
- Message display
- Configuration window
- etc.

(>>> *4.2 "KUKA smartHMI user interface" Page 61*)

Depending on customer-specific settings, the user interface may vary from the standard interface.

2.3 System requirements

The System Software 8.5 can be run on the following robot controller:

- KR C4
- with Windows Embedded Standard 7 V4.x
- and with at least 2 GB RAM

2.4 Intended use of the KUKA System Software

Use

The KUKA System Software is intended exclusively for the operation of a KUKA industrial robot or customer-specific kinematic system.

Each version of the KUKA System Software may be operated exclusively in accordance with the specified system requirements.

Misuse

Any use or application deviating from the intended use is deemed to be misuse and is not allowed. KUKA Deutschland GmbH is not liable for any damage resulting from such misuse. The risk lies entirely with the user.

Examples of such misuse include:

- Operation of a kinematic system that is neither a KUKA industrial robot nor a customer-specific kinematic system
- Operation of the KSS not in accordance with the specified system requirements

2.5 KUKA USB sticks

Special USB sticks from KUKA are available for the KR C4 and VKR C4 robot controllers. The sticks are available in the following variants:

- Bootable variant, exclusively in conjunction with the software KUKA.RecoveryUSB

  Color: Gray or black
- Non-bootable variant for data backup

  Color: Orange

For further information about the USB sticks, please contact KUKA Deutschland GmbH.

NOTICE

For activities on the robot controller requiring a USB stick, use of a KUKA stick is generally recommended.

- The KUKA sticks are validated for use with the KR C4/VKR C4.
- Data may be lost if sticks from other manufacturers are used.

3 Safety

3.1 General

3.1.1 Liability

The device described in this document is either an industrial robot or a component thereof.

Components of the industrial robot:

- Manipulator
- Robot controller
- Teach pendant
- Connecting cables
- External axes (optional)
  e.g. linear unit, turn-tilt table, positioner
- Software
- Options, accessories

The industrial robot is built using state-of-the-art technology and in accordance with the recognized safety rules. Nevertheless, misuse of the industrial robot may constitute a risk to life and limb or cause damage to the industrial robot and to other material property.

The industrial robot may only be used in perfect technical condition in accordance with its designated use and only by safety-conscious persons who are fully aware of the risks involved in its operation. Use of the industrial robot is subject to compliance with this document and with the declaration of incorporation supplied together with the industrial robot. Any functional disorders affecting safety must be rectified immediately.

Safety information

Information about safety may not be construed against KUKA Deutschland GmbH. Even if all safety instructions are followed, this is not a guarantee that the industrial robot will not cause personal injuries or material damage.

No modifications may be carried out to the industrial robot without the authorization of KUKA Deutschland GmbH. Additional components (tools, software, etc.), not supplied by KUKA Deutschland GmbH, may be integrated into the industrial robot. The user is liable for any damage these components may cause to the industrial robot or to other material property.

In addition to the Safety chapter, this document contains further safety instructions. These must also be observed.

3.1.2 Intended use of the industrial robot

The industrial robot is intended exclusively for the use designated in the “Purpose” chapter of the operating instructions or assembly instructions.

Any use or application deviating from the intended use is deemed to be misuse and is not allowed. The manufacturer is not liable for any damage resulting from such misuse. The risk lies entirely with the user.

Operation of the industrial robot in accordance with its intended use also requires compliance with the operating and assembly instructions for the

individual components, with particular reference to the maintenance specifications.

Misuse

Any use or application deviating from the intended use is deemed to be misuse and is not allowed. This includes e.g.:

- Use as a climbing aid
- Operation outside the specified operating parameters
- Operation without the required safety equipment

3.1.3 EC declaration of conformity and declaration of incorporation

The industrial robot constitutes partly completed machinery as defined by the EC Machinery Directive. The industrial robot may only be put into operation if the following preconditions are met:

- The industrial robot is integrated into a complete system.

  or: The industrial robot, together with other machinery, constitutes a complete system.

  or: All safety functions and safeguards required for operation in the complete machine as defined by the EC Machinery Directive have been added to the industrial robot.
- The complete system complies with the EC Machinery Directive. This has been confirmed by means of a conformity assessment procedure.

EC declaration of conformity

The system integrator must issue an EC declaration of conformity for the complete system in accordance with the Machinery Directive. The EC declaration of conformity forms the basis for the CE mark for the system. The industrial robot must always be operated in accordance with the applicable national laws, regulations and standards.

The robot controller has a CE mark in accordance with the EMC Directive and the Low Voltage Directive.

Declaration of incorporation

The partly completed machinery is supplied with a declaration of incorporation in accordance with Annex II B of the Machinery Directive 2006/42/EC. The assembly instructions and a list of essential requirements complied with in accordance with Annex I are integral parts of this declaration of incorporation.

The declaration of incorporation declares that the start-up of the partly completed machinery is not allowed until the partly completed machinery has been incorporated into machinery, or has been assembled with other parts to form machinery, and this machinery complies with the terms of the EC Machinery Directive, and the EC declaration of conformity is present in accordance with Annex II A.

3.1.4 Terms used

STOP 0, STOP 1 and STOP 2 are the stop definitions according to EN 60204-1:2006.

Term	Description
Axis range	Range of each axis, in degrees or millimeters, within which it may move. The axis range must be defined for each axis.
Stopping distance	Stopping distance = reaction distance + braking distance The stopping distance is part of the danger zone.
Workspace	Area within which the robot may move. The workspace is derived from the individual axis ranges.
User	The user of the industrial robot can be the management, employer or delegated person responsible for use of the industrial robot.
Danger zone	The danger zone consists of the workspace and the stopping distances of the manipulator and external axes (optional).
Service life	The service life of a safety-relevant component begins at the time of delivery of the component to the customer. The service life is not affected by whether the component is used or not, as safety-relevant components are also subject to aging during storage.
KUKA smartPAD	see “smartPAD”
KUKA smartPAD-2	
Manipulator	The robot arm and the associated electrical installations
Safety zone	The safety zone is situated outside the danger zone.
Safe operational stop	The safe operational stop is a standstill monitoring function. It does not stop the robot motion, but monitors whether the robot axes are stationary. If these are moved during the safe operational stop, a safety stop STOP 0 is triggered. The safe operational stop can also be triggered externally. When a safe operational stop is triggered, the robot controller sets an output to the field bus. The output is set even if not all the axes were stationary at the time of triggering, thereby causing a safety stop STOP 0 to be triggered.
Safety STOP 0	A stop that is triggered and executed by the safety controller. The safety controller immediately switches off the drives and the power supply to the brakes. **Note:** This stop is called safety STOP 0 in this document.
Safety STOP 1	A stop that is triggered and monitored by the safety controller. The braking operation is carried out by the non-safety-oriented section of the robot controller and monitored by the safety controller. As soon as the manipulator has stopped, the safety controller deactivates the drives and the power supply of the brakes. When a safety STOP 1 is triggered, the robot controller sets an output to the field bus. The safety STOP 1 can also be triggered externally. **Note:** This stop is called safety STOP 1 in this document.

Term	Description
Safety STOP 2	A stop that is triggered and monitored by the safety controller. The braking operation is carried out by the non-safety-oriented section of the robot controller and monitored by the safety controller. The drives remain activated and the brakes released. As soon as the manipulator is at a standstill, a safe operational stop is triggered. When a safety STOP 2 is triggered, the robot controller sets an output to the field bus. The safety STOP 2 can also be triggered externally. **Note:** This stop is called safety STOP 2 in this document.
Safety options	Generic term for options which make it possible to configure additional safe monitoring functions in addition to the standard safety functions. Example: SafeOperation
smartPAD	Programming device for the robot controller The smartPAD has all the operator control and display functions required for operating and programming the industrial robot. Two models exist: • smartPAD • smartPAD-2 In turn, for every model there are variants (e.g. with the varying lengths of connecting cables). The designation “KUKA smartPAD” or “smartPAD” refers to both models unless an explicit distinction is made.
Stop category 0	The drives are deactivated immediately and the brakes are applied. The manipulator and any external axes (optional) perform path-oriented braking. **Note:** This stop category is called STOP 0 in this document.
Stop category 1	The manipulator and any external axes (optional) perform path-maintaining braking. • Operating mode T1: The drives are deactivated as soon as the robot has stopped, but no later than after 680 ms. • Operating modes T2, AUT (KR C4), AUT EXT (KR C4), EXT (VKR C4): The drives are switched off after 1.5 s. **Note:** This stop category is called STOP 1 in this document.
Stop category 1 - Drive Ramp Stop	The manipulator and any external axes (optional) perform path-oriented braking. • Operating mode T1: The drives are deactivated as soon as the robot has stopped, but no later than after 680 ms. • Operating modes T2, AUT (KR C4), AUT EXT (KR C4), EXT (VKR C4): The drives are switched off after 1.5 s. **Note:** This stop category is called STOP 1 - DRS in this document.
Stop category 2	The drives are not deactivated and the brakes are not applied. The manipulator and any external axes (optional) are braked with a path-maintaining braking ramp. **Note:** This stop category is called STOP 2 in this document.

Term	Description
System integrator (plant integrator)	The system integrator is responsible for safely integrating the industrial robot into a complete system and commissioning it.
T1	Test mode, Manual Reduced Velocity (<= 250 mm/s)
T2	Test mode, Manual High Velocity (> 250 mm/s permissible)
External axis	Motion axis which is not part of the manipulator but which is controlled using the robot controller, e.g. KUKA linear unit, turn-tilt table, Posiflex.

3.2 Personnel

The following persons or groups of persons are defined for the industrial robot:

- User
- Personnel

All persons working with the industrial robot must have read and understood the industrial robot documentation, including the safety chapter.

User

The user must observe the labor laws and regulations. This includes e.g.:

- The user must comply with his monitoring obligations.
- The user must carry out briefing at defined intervals.

Personnel

Personnel must be instructed, before any work is commenced, in the type of work involved and what exactly it entails as well as any hazards which may exist. Instruction must be carried out regularly. Instruction is also required after particular incidents or technical modifications.

Personnel includes:

- System integrator
- Operators, subdivided into:
  - Start-up, maintenance and service personnel
  - Operating personnel
  - Cleaning personnel

Installation, exchange, adjustment, operation, maintenance and repair must be performed only as specified in the operating or assembly instructions for the relevant component of the industrial robot and only by personnel specially trained for this purpose.

System integrator

The industrial robot is safely integrated into a complete system by the system integrator.

The system integrator is responsible for the following tasks:

- Installing the industrial robot
- Connecting the industrial robot
- Performing risk assessment
- Implementing the required safety functions and safeguards

- Issuing the EC declaration of conformity
- Attaching the CE mark
- Creating the operating instructions for the system

Operators

The operator must meet the following preconditions:

- The operator must be trained for the work to be carried out.
- Work on the system must only be carried out by qualified personnel. These are people who, due to their specialist training, knowledge and experience, and their familiarization with the relevant standards, are able to assess the work to be carried out and detect any potential hazards.

Work on the electrical and mechanical equipment of the industrial robot may only be carried out by specially trained personnel.

3.3 Workspace, safety zone and danger zone

Workspaces are to be restricted to the necessary minimum size. A workspace must be safeguarded using appropriate safeguards.

The safeguards (e.g. safety gate) must be situated inside the safety zone. In the case of a stop, the manipulator and external axes (optional) are braked and come to a stop within the danger zone.

The danger zone consists of the workspace and the stopping distances of the manipulator and external axes (optional). It must be safeguarded by means of physical safeguards to prevent danger to persons or the risk of material damage.

3.3.1 Determining stopping distances

The system integrator's risk assessment may indicate that the stopping distances must be determined for an application. In order to determine the stopping distances, the system integrator must identify the safety-relevant points on the programmed path.

When determining the stopping distances, the robot must be moved with the tool and loads which are also used in the application. The robot must be at operating temperature. This is the case after approx. 1 h in normal operation.

During execution of the application, the robot must be stopped at the point from which the stopping distance is to be calculated. This process must be repeated several times with a safety stop 0 and a safety stop 1. The least favorable stopping distance is decisive.

A safety stop 0 can be triggered by a safe operational stop via the safety interface, for example. If a safety option is installed, it can be triggered, for instance, by a space violation (e.g. the robot exceeds the limit of an activated workspace in Automatic mode).

A safety stop 1 can be triggered by pressing the EMERGENCY STOP device on the smartPAD, for example.

3.4 Triggers for stop reactions

Stop reactions of the industrial robot are triggered in response to operator actions or as a reaction to monitoring functions and error messages. The

following table shows the different stop reactions according to the operating mode that has been set.

Trigger	T1, T2	AUT, AUT EXT
Start key released	STOP 2	-
STOP key pressed	STOP 2	
Drives OFF	STOP 1	
$MOVE_ENABLE input drops out	STOP 2	
Power switched off via main switch or power failure	STOP 0	
Internal error in non-safety-oriented part of the robot controller	STOP 0 or STOP 1 (dependent on the cause of the error)	
Operating mode changed during operation	Safety stop 2	
Safety gate opened (operator safety)	-	Safety stop 1
Enabling switch released	Safety stop 2	-
Enabling switch pressed fully down or error	Safety stop 1	-
E-STOP pressed	Safety stop 1	
Error in safety controller or periphery of the safety controller	Safety stop 0	

3.5 Safety functions

3.5.1 Overview of the safety functions

The following safety functions are present in the industrial robot:

- Selecting the operating mode
- Operator safety (= connection for the monitoring of physical safeguards)
- EMERGENCY STOP device
- Enabling device
- External safe operational stop
- External safety stop 1
- External safety stop 2
- Velocity monitoring in T1

The safety functions of the industrial robot meet the following requirements:

- **Category 3** and **Performance Level d** in accordance with EN ISO 13849-1

The requirements are only met on the following condition, however:

- The EMERGENCY STOP device is pressed at least once every 12 months.

The following components are involved in the safety functions:

- Safety controller in the control PC
- KUKA smartPAD
- Cabinet Control Unit (CCU)
- Resolver Digital Converter (RDC)
- KUKA Power Pack (KPP)
- KUKA Servo Pack (KSP)
- Safety Interface Board (SIB) (if used)

There are also interfaces to components outside the industrial robot and to other robot controllers.

DANGER

In the absence of operational safety functions and safeguards, the industrial robot can cause personal injury or material damage. If safety functions or safeguards are dismantled or deactivated, the industrial robot may not be operated.

During system planning, the safety functions of the overall system must also be planned and designed. The industrial robot must be integrated into this safety system of the overall system.

3.5.2 Safety controller

The safety controller is a unit inside the control PC. It links safety-relevant signals and safety-relevant monitoring functions.

Safety controller tasks:

- Switching off the drives; applying the brakes
- Monitoring the braking ramp
- Standstill monitoring (after the stop)
- Velocity monitoring in T1
- Evaluation of safety-relevant signals
- Setting of safety-oriented outputs

3.5.3 Selecting the operating mode

Operating modes

The industrial robot can be operated in the following modes:

- Manual Reduced Velocity (T1)
- Manual High Velocity (T2)
- Automatic (AUT)
- Automatic External (AUT EXT)

Do not change the operating mode while a program is running. If the operating mode is changed during program execution, the industrial robot is stopped with a safety stop 2.

Operating mode	Use	Velocities
T1	For test operation, programming and teaching	• Program verification: Programmed velocity, maximum 250 mm/s • Jog mode: Jog velocity, maximum 250 mm/s
T2	For test operation	• Program verification: Programmed velocity • Jog mode: Not possible
AUT	For industrial robots without higher-level controllers	• Program operation: Programmed velocity • Jog mode: Not possible
AUT EXT	For industrial robots with higher-level controllers, e.g. PLC	• Program operation: Programmed velocity • Jog mode: Not possible

Mode selector switch

The user can change the operating mode via the connection manager. The connection manager is a view that is called by means of the mode selector switch on the smartPAD.

The mode selector switch may be one of the following variants:

- With key

  It is only possible to change operating mode if the key is inserted.
- Without key

WARNING

If the smartPAD is fitted with a mode selector switch without a key:
An additional device must be present to ensure that the relevant functions cannot be executed by all users, but only by a restricted group of people.
The device itself must not trigger motions of the industrial robot or other hazards. If this device is missing, death or severe injuries may result.

The system integrator is responsible for ensuring that such a device is implemented.

3.5.4 “Operator safety” signal

The “operator safety” signal is used for monitoring physical safeguards, e.g. safety gates. Automatic operation is not possible without this signal. In the event of a loss of signal during automatic operation (e.g. safety gate is opened), the manipulator stops with a safety stop 1.

Operator safety is not active in modes T1 (Manual Reduced Velocity) and T2 (Manual High Velocity).

WARNING

Following a loss of signal, automatic operation may only be resumed when the safeguard has been closed and when the closing has been acknowledged. This acknowledgement is to prevent automatic operation from being resumed inadvertently while there are still persons in the danger zone, e.g. due to the safety gate closing accidentally.
The acknowledgement must be designed in such a way that an actual check of the danger zone can be carried out first. Other acknowledgement functions (e.g. an acknowlegement which is automatically triggered by closure of the safeguard) are not permitted.
The system integrator is responsible for ensuring that these criteria are met. Failure to met them may result in death, severe injuries or considerable damage to property.

3.5.5 EMERGENCY STOP device

The EMERGENCY STOP device for the industrial robot is the EMERGENCY STOP device on the smartPAD. The device must be pressed in the event of a hazardous situation or emergency.

Reactions of the industrial robot if the EMERGENCY STOP device is pressed:

- The manipulator and any external axes (optional) are stopped with a safety stop 1.

Before operation can be resumed, the EMERGENCY STOP device must be turned to release it.

WARNING

Tools and other equipment connected to the robot must be integrated into the EMERGENCY STOP circuit on the system side if they could constitute a potential hazard.
Failure to observe this precaution may result in death, severe injuries or considerable damage to property.

There must always be at least one external EMERGENCY STOP device installed. This ensures that an EMERGENCY STOP device is available even when the smartPAD is disconnected.

(>>> *3.5.7 "External EMERGENCY STOP device" Page 33*)

3.5.6 Logging off from the higher-level safety controller

If the robot controller is connected to a higher-level safety controller, this connection will inevitably be terminated in the following cases:

- Switching off the voltage via the main switch of the robot
  Or power failure
- Shutdown of the robot controller via the smartHMI
- Activation of a WorkVisual project in WorkVisual or directly on the robot controller
- Changes to **Start-up** > **Network configuration**
- Changes to **Configuration** > **Safety configuration**
- **I/O drivers** > **Reconfigure**
- Restoration of an archive

Effect of the interruption:

- If a discrete safety interface is used, this triggers an EMERGENCY STOP for the overall system.
- If the Ethernet interface is used, the KUKA safety controller generates a signal that prevents the higher-level controller from triggering an EMERGENCY STOP for the overall system.

If the Ethernet safety interface is used: In his risk assessment, the system integrator must take into consideration whether the fact that switching off the robot controller does not trigger an EMERGENCY STOP of the overall system could constitute a hazard and, if so, how this hazard can be countered.
Failure to take this into consideration may result in death, injuries or damage to property.

WARNING

If a robot controller is switched off, the E-STOP device on the smartPAD is no longer functional. The user is responsible for ensuring that the smartPAD is either covered or removed from the system. This serves to prevent operational and non-operational EMERGENCY STOP devices from becoming interchanged.
Failure to observe this precaution may result in death, injuries or damage to property.

3.5.7 External EMERGENCY STOP device

Every operator station that can initiate a robot motion or other potentially hazardous situation must be equipped with an EMERGENCY STOP device. The system integrator is responsible for ensuring this.

There must always be at least one external EMERGENCY STOP device installed. This ensures that an EMERGENCY STOP device is available even when the smartPAD is disconnected.

External EMERGENCY STOP devices are connected via the customer interface. External EMERGENCY STOP devices are not included in the scope of supply of the industrial robot.

3.5.8 Enabling device

The enabling devices of the industrial robot are the enabling switches on the smartPAD.

- **smartPAD**: 3 enabling switches
- **smartPAD-2**: 4 enabling switches

The enabling switches have 3 positions:

- Not pressed
- Center position
- Fully pressed (panic position)

In the test modes, the manipulator can only be moved if at least one of the enabling switches is held in the center position.

It is possible to hold several enabling switches in the center position simultaneously. This makes it possible to adjust grip from one enabling switch to another one.

In the test modes, the manipulator can be stopped in the following ways:

- Press at least one enabling switch down fully.

  Pressing an enabling switch down fully triggers a safety stop 1.

- Or release all enabling switches.

  Releasing all (!) enabling switches held in the center position triggers a safety stop 2.

Releasing one of multiple enabling switches held in the center position does not trigger a reaction.
If multiple switches are held in the center position, it is not possible to distinguish whether one of them was intentionally released or if it was unintentionally released as the result of an accident.

If an enabling switch malfunctions (e.g. jams in the center position), the industrial robot can be stopped using one of the following methods:

- Press another enabling switch down fully.
- Actuate the EMERGENCY STOP device.
- Release the Start key.

WARNING

The enabling switches must not be held down by adhesive tape or other means or tampered with in any other way.
Death, injuries or damage to property may result.

Function test

The function of the enabling switches must be tested in the following cases:

- Following initial start-up or recommissioning of the industrial robot
- After a software update
- After disconnecting and reconnecting a smartPAD (the same smartPAD or another one)
- The test must be carried out at least once every 12 months.

To test, perform the following steps separately for each enabling switch:

1. Move the manipulator in a test mode.
2. While the manipulator is moving, press the enabling switch down fully and hold it down for 3 seconds.

The test is passed in the following case:

- The manipulator stops.
- And: No error message for the enabling device is displayed (*Enabling switch error* or similar).

If the test has not been passed for one or more enabling switches, the smartPAD must be exchanged and the test must be performed again.

3.5.9 External enabling device

External enabling devices are required if it is necessary for more than one person to be in the danger zone of the industrial robot.

The function of the external enabling switches must be tested at least once every 12 months.

Which interface can be used for connecting external enabling devices is described in the “Planning” chapter of the robot controller operating instructions and assembly instructions.

External enabling devices are not included in the scope of supply of the industrial robot.

3.5.10 External safe operational stop

The safe operational stop can be triggered via an input on the customer interface. The state is maintained as long as the external signal is FALSE. If the external signal is TRUE, the manipulator can be moved again. No acknowledgement is required.

3.5.11 External safety stop 1 and external safety stop 2

Safety stop 1 and safety stop 2 can be triggered via an input on the customer interface. The state is maintained as long as the external signal is FALSE. If the external signal is TRUE, the manipulator can be moved again. No acknowledgement is required.

If interface X11 is selected as the customer interface, only the signal **Safety stop 2** is available.

3.5.12 Velocity monitoring in T1

In T1 mode, the velocity is safely monitored on both an axis-specific and Cartesian basis.

- Applicable up to and including System Software 8.5.7:
  For customer kinematic systems (CKs), only the axis-specific velocity is safely monitored in T1 mode. The Cartesian position on the flange is only available for KUKA kinematic systems using safe technology.
- Applicable for System Software 8.5.8 or higher:
  For all kinematic systems, the axis-specific velocity is safely monitored in T1 mode. The Cartesian velocity is safely monitored in T1 if the associated setting is activated in the safety configuration.

Axis-specific monitoring

If an axis exceeds its velocity limit, a safety stop 0 is triggered.

- Default limit value for rotational axes: 30 °/s
- Default limit value for linear axes: 250 mm/s

From System Software 8.3 onwards, the axis-specific monitoring can be configured using the **Maximum velocity T1** parameter. Further information about this can be found in the **Operating and Programming Instructions for System Integrators** documentation for the System Software.

Cartesian monitoring

The Cartesian monitoring refers to the velocity at the flange. If a limit value is exceeded, a safety stop 0 is triggered.

- Default limit value: 250 mm/s

If an additional safety option (e.g. SafeOperation) is used, the limit value can be configured. It can be reduced, but not increased.

3.6 Additional protective equipment

3.6.1 Jog mode

In the operating modes T1 (Manual Reduced Velocity) and T2 (Manual High Velocity), the robot controller can only execute programs in jog mode. This means that it is necessary to hold down an enabling switch and the Start key in order to execute a program.

- Releasing the enabling switch triggers a safety stop 2.
- Pressing the enabling switch down fully (panic position) triggers a safety stop 1.
- Releasing the Start key triggers a STOP 2.

3.6.2 Software limit switches

The axis ranges of all manipulator and positioner axes are limited by means of adjustable software limit switches. These software limit switches only serve as machine protection and must be adjusted in such a way that the manipulator/positioner cannot hit the mechanical end stops.

The software limit switches are set during commissioning of an industrial robot.

Further information is contained in the operating and programming instructions.

3.6.3 Mechanical end stops

Depending on the robot variant, the axis ranges of the main and wrist axes of the manipulator are partially limited by mechanical end stops.

Additional mechanical end stops can be installed on the external axes.

WARNING

If the manipulator or an external axis hits an obstruction or a mechanical end stop or mechanical axis limitation, the manipulator can no longer be operated safely. The manipulator must be taken out of operation and KUKA Deutschland GmbH must be consulted before it is put back into operation.

3.6.4 Mechanical axis limitation (optional)

Some manipulators can be fitted with mechanical axis limitation systems in axes A1 to A3. The axis limitation systems restrict the working range to the required minimum. This increases personal safety and protection of the system.

In the case of manipulators that are not designed to be fitted with mechanical axis limitation, the workspace must be laid out in such a way that there is no danger to persons or material property, even in the absence of mechanical axis limitation.

If this is not possible, the workspace must be limited by means of photoelectric barriers, photoelectric curtains or obstacles on the system side. There must be no shearing or crushing hazards at the loading and transfer areas.

This option is not available for all robot models. Information on specific robot models can be obtained from KUKA Deutschland GmbH.

3.6.5 Options for moving the manipulator without drive energy

The system user is responsible for ensuring that the training of personnel with regard to the response to emergencies or exceptional situations also includes how the manipulator can be moved without drive energy.

Description

The following options are available for moving the manipulator without drive energy after an accident or malfunction:

- Release device (optional)

  The release device can be used for the main axis drive motors and, depending on the robot variant, also for the wrist axis drive motors.
- Brake release device (option)

  The brake release device is designed for robot variants whose motors are not freely accessible.
- Moving the wrist axes directly by hand

  There is no release device available for the wrist axes of variants in the low payload category. This is not necessary because the wrist axes can be moved directly by hand.

Information about the options available for the various robot models and about how to use them can be found in the assembly and operating instructions for the robot or requested from KUKA Deutschland GmbH.

NOTICE

Moving the manipulator without drive energy can damage the motor brakes of the axes concerned. The motor must be replaced if the brake has been damaged. The manipulator may therefore be moved without drive energy only in emergencies, e.g. for rescuing persons.

3.6.6 Labeling on the industrial robot

All plates, labels, symbols and marks constitute safety-relevant parts of the industrial robot. They must not be modified or removed.

Labeling on the industrial robot consists of:

- Identification plates
- Warning signs
- Safety symbols
- Designation labels
- Cable markings
- Rating plates

Further information is contained in the technical data of the operating instructions or assembly instructions of the components of the industrial robot.

3.6.7 External safeguards

The access of persons to the danger zone of the industrial robot must be prevented by means of safeguards. It is the responsibility of the system integrator to ensure this.

Physical safeguards must meet the following requirements:

- They meet the requirements of EN ISO 14120.
- They prevent access of persons to the danger zone and cannot be easily circumvented.
- They are sufficiently fastened and can withstand all forces that are likely to occur in the course of operation, whether from inside or outside the enclosure.
- They do not, themselves, represent a hazard or potential hazard.
- Prescribed clearances, e.g. to danger zones, are adhered to.

Safety gates (maintenance gates) must meet the following requirements:

- They are reduced to an absolute minimum.
- The interlocks (e.g. safety gate switches) are linked to the operator safety input of the robot controller via safety gate switching devices or safety PLC.
- Switching devices, switches and the type of switching conform to the requirements of Performance Level d and category 3 according to EN ISO 13849-1.
- Depending on the risk situation: the safety gate is additionally safeguarded by means of a locking mechanism that only allows the gate to be opened if the manipulator is safely at a standstill.
- The button for acknowledging the safety gate is located outside the space limited by the safeguards.

Further information is contained in the corresponding standards and regulations. These also include EN ISO 14120.

Other safety equipment

Other safety equipment must be integrated into the system in accordance with the corresponding standards and regulations.

3.7 Overview of operating modes and safety functions

The following table indicates the operating modes in which the safety functions are active.

Safety functions	T1	T2	AUT	AUT EXT
Operator safety	-	-	Active	Active
EMERGENCY STOP device	Active	Active	Active	Active
Enabling device	Active	Active	-	-
Reduced velocity during program verification	Active	-	-	-
Jog mode	Active	Active	-	-
Software limit switches	Active	Active	Active	Active

3.8 Safety measures

3.8.1 General safety measures

The industrial robot may only be used in perfect technical condition in accordance with its intended use and only by safety-conscious persons. Operator errors can result in personal injury and damage to property.

It is important to be prepared for possible movements of the industrial robot even after the robot controller has been switched off and locked out. Incorrect installation (e.g. overload) or mechanical defects (e.g. brake defect) can cause the manipulator or external axes to sag. If work is to be carried out on a switched-off industrial robot, the manipulator and external axes must first be moved into a position in which they are unable to move on their own, whether the payload is mounted or not. If this is not possible, the manipulator and external axes must be secured by appropriate means.

DANGER

In the absence of operational safety functions and safeguards, the industrial robot can cause personal injury or material damage. If safety functions or safeguards are dismantled or deactivated, the industrial robot may not be operated.

DANGER

Standing underneath the robot arm can cause death or injuries. For this reason, standing underneath the robot arm is prohibited!

CAUTION

The motors reach temperatures during operation which can cause burns to the skin. Contact must be avoided. Appropriate safety precautions must be taken, e.g. protective gloves must be worn.

smartPAD

The user must ensure that the industrial robot is only operated with the smartPAD by authorized persons.

If more than one smartPAD is used in the overall system, it must be ensured that it is clearly recognizable which smartPAD is connected to which industrial robot. They must not be interchanged.

WARNING

The operator must ensure that decoupled smartPADs are immediately removed from the system and stored out of sight and reach of personnel working on the industrial robot. This serves to prevent operational and non-operational EMERGENCY STOP devices from becoming interchanged.
Failure to observe this precaution may result in death, severe injuries or considerable damage to property.

In certain cases, and at least every 12 months, the enabling switches on the smartPAD must be subjected to a function test.

(>>> *"Function test" Page 34*)

Modifications

After modifications to the industrial robot, checks must be carried out to ensure the required safety level. The valid national or regional work safety regulations must be observed for this check. The correct functioning of all safety functions must also be tested.

New or modified programs must always be tested first in Manual Reduced Velocity mode (T1).

After modifications to the industrial robot, existing programs must always be tested first in Manual Reduced Velocity mode (T1). This applies to all components of the industrial robot and includes e.g. modifications of the external axes or to the software and configuration settings.

Faults

The following tasks must be carried out in the case of faults in the industrial robot:

- Switch off the robot controller and secure it (e.g. with a padlock) to prevent unauthorized persons from switching it on again.
- Indicate the fault by means of a label with a corresponding warning (tagout).
- Keep a record of the faults.
- Eliminate the fault and carry out a function test.

3.8.2 IT security

The industrial robot may only be used in perfect technical condition in accordance with its intended use and only by safety-conscious persons.

In particular, security-conscious use includes that it be operated in an IT environment which meets the current security-relevant standards and for which there is an overall concept for IT security.

IT security entails not only technical aspects but, at a minimum, also those of organization, personnel and infrastructure.
KUKA recommends that operators immediately implement an information security management system for their products which designs, coordinates and monitors the tasks related to information security.

Sources for information about IT security for companies include:

- Independent consulting firms
- National cyber security authorities

National authorities often make their recommendations available on the Internet. In addition to their official language, some national authorities provide their information in English.

3.8.3 Transportation

Manipulator

The prescribed transport position of the manipulator must be observed. Transportation must be carried out in accordance with the operating instructions or assembly instructions of the robot.

Avoid vibrations and impacts during transportation in order to prevent damage to the manipulator.

Robot controller

The prescribed transport position of the robot controller must be observed. Transportation must be carried out in accordance with the operating instructions or assembly instructions of the robot controller.

Avoid vibrations and impacts during transportation in order to prevent damage to the robot controller.

External axis (optional)

The prescribed transport position of the external axis (e.g. KUKA linear unit, turn-tilt table, positioner) must be observed. Transportation must be carried out in accordance with the operating instructions or assembly instructions of the external axis.

3.8.4 Start-up and recommissioning

Before starting up systems and devices for the first time, a check must be carried out to ensure that the systems and devices are complete and operational, that they can be operated safely and that any damage is detected.

The valid national or regional work safety regulations must be observed for this check. The correct functioning of all safety functions must also be tested.

The passwords for the user groups must be changed in the KUKA System Software before start-up. The passwords must only be communicated to authorized personnel.

WARNING

The robot controller is preconfigured for the specific industrial robot. If cables are interchanged, the manipulator and the external axes (optional) may receive incorrect data and can thus cause personal injury or material damage. If a system consists of more than one manipulator, always connect the connecting cables to the manipulators and their corresponding robot controllers.

If additional components (e.g. cables), which are not part of the scope of supply of KUKA Deutschland GmbH, are integrated into the industrial robot, the user is responsible for ensuring that these components do not adversely affect or disable safety functions.

NOTICE

If the internal cabinet temperature of the robot controller differs greatly from the ambient temperature, condensation can form, which may cause damage to the electrical components. Do not put the robot controller into operation until the internal temperature of the cabinet has adjusted to the ambient temperature.

Function test

The following tests must be carried out before start-up and recommissioning:

General test:

It must be ensured that:

- The industrial robot is correctly installed and fastened in accordance with the specifications in the documentation.
- There is no damage to the robot that could be attributed to external forces. Examples: Dents or abrasion that could be caused by an impact or collision.

  WARNING

  In the case of such damage, the affected components must be exchanged. In particular, the motor and counterbalancing system must be checked carefully.
  External forces can cause non-visible damage. For example, it can lead to a gradual loss of drive power from the motor, resulting in unintended movements of the manipulator. Death, injuries or considerable damage to property may otherwise result.

- There are no foreign bodies or loose parts on the industrial robot.
- All required safety equipment is correctly installed and operational.

- The power supply ratings of the industrial robot correspond to the local supply voltage and mains type.
- The ground conductor and the equipotential bonding cable are sufficiently rated and correctly connected.
- The connecting cables are correctly connected and the connectors are locked.

Test of the safety functions:

A function test must be carried out for the following safety functions to ensure that they are functioning correctly:

- Local EMERGENCY STOP device
- External EMERGENCY STOP device (input and output)
- Enabling device (in the test modes)
- Operator safety
- All other safety-relevant inputs and outputs used
- Other external safety functions

3.8.4.1 Checking machine data and safety configuration

WARNING

The industrial robot must not be moved if incorrect machine data or an incorrect controller configuration are loaded. Death, severe injuries or considerable damage to property may otherwise result. The correct data must be loaded.

- Following the start-up procedure, the practical tests for the machine data must be carried out. The tool must be calibrated (either via an actual calibration or through numerical entry of the data).
- Following modifications to the machine data, the safety configuration must be checked.
- After activation of a WorkVisual project on the robot controller, the safety configuration must be checked.
- If machine data are adopted when checking the safety configuration (regardless of the reason for the safety configuration check), the practical tests for the machine data must be carried out.
- System Software 8.3 or higher: If the checksum or the activation code of the safety configuration has changed, the safe axis monitoring functions must be checked.

  Up to and including System Software 8.5, the value in the safety configuration is “Checksum”; for 8.6 and above, it is “Activation code”.

Information about checking the safety configuration and the safe axis monitoring functions is contained in the Operating and Programming Instructions for System Integrators.

If the practical tests are not successfully completed in the initial start-up, KUKA Deutschland GmbH must be contacted.

If the practical tests are not successfully completed during a different procedure, the machine data and the safety-relevant controller configuration must be checked and corrected.

General practical test

If practical tests are required for the machine data, this test must always be carried out.

For 6-axis robots:

The following methods are available for performing the practical test:

- TCP calibration with the XYZ 4-point method

  The practical test is passed if the TCP has been successfully calibrated.

Or:

1. Align the TCP with a freely selected point. The point serves as a reference point.
   - The point must be located so that reorientation is possible.
   - The point must not be located on the Z axis of the FLANGE coordinate system.
2. Move the TCP manually at least 45° once in each of the A, B and C directions.

   The movements do not have to be accumulative, i.e. after motion in one direction it is possible to return to the original position before moving in the next direction.

   The practical test is passed if the TCP does not deviate from the reference point by more than 2 cm in total.

For palletizing robots:

Palletizing robots, in this case, are either robots that can be used only as palletizers from the start or robots operated in palletizing mode. The latter must also be in palletizing mode during the practical test.

First part:

1. Mark the starting position of the TCP.

   Also read and note the starting position from the **Actual position – Cartesian** display on the smartHMI.
2. Jog the TCP in the X direction. The distance must be at least 20% of the robot's maximum reach. Determine the exact length via the **Actual position** display.
3. Measure the distance covered and compare it with the distance value displayed on the smartHMI. The deviation must be < 5%.
4. Repeat steps 1 and 2 for the Y direction and Z direction.

The first part of the practical test is passed if the deviation is < 5% in every direction.

Second part:

- Rotate the tool manually about A by 45°: once in the plus direction, once in the minus direction. At the same time, observe the TCP.

The second part of the practical test is passed if the position of the TCP in space is not altered during the rotations.

Practical test for axes that are not mathematically coupled

If practical tests are required for the machine data, this test must be carried out when axes are present that are not mathematically coupled.

1. Mark the starting position of the axis that is not mathematically coupled.

   Also read and note the start position from the **Actual position** display on the smartHMI.
2. Move the axis manually by a freely selected path length. Determine the path length from the **Actual position** display.
   - Move linear axes a specific distance.
   - Move rotational axes through a specific angle.

3. Measure the length of the path covered and compare it with the value displayed on the smartHMI.

   The practical test is passed if the values differ by no more than 5%.
4. Repeat the test for each axis that is not mathematically coupled.

Practical test for robot on KUKA linear unit

If practical tests are required for the machine data, this test must be carried out if the robot and KL are mathematically coupled.

- Move the KL manually in Cartesian mode.

  The practical test is passed if the TCP does not move at the same time.

Practical test for couplable axes

If practical tests are required for the machine data, this test must be carried out when axes are present that can be physically coupled and uncoupled, e.g. a servo gun.

1. Physically uncouple the couplable axis.
2. Move all the remaining axes individually.

   The practical test is passed if it has been possible to move all the remaining axes.

3.8.4.2 Start-up mode

Description

The industrial robot can be set to Start-up mode via the smartHMI user interface. In this mode, the manipulator can be moved in T1 without the external safeguards being put into operation.

The safety interface used affects “Start-up” mode:

Discrete safety interface

- System Software 8.2 or earlier:

  Start-up mode is always possible if all input signals at the discrete safety interface have the state “logic zero”. If this is not the case, the robot controller prevents or terminates Start-up mode.

  If an additional discrete safety interface for safety options is used, the inputs there must also have the state “logic zero”.
- System Software 8.3 or higher:

  Start-up mode is always possible. This also means that it is independent of the state of the inputs at the discrete safety interface.

  If an additional discrete safety interface is used for safety options: The states of these inputs are also irrelevant.

Ethernet safety interface

The robot controller prevents or terminates Start-up mode if a connection to a higher-level safety system exists or is established.

Effect

When the Start-up mode is activated, all outputs are automatically set to the state “logic zero”.

If the robot controller has a peripheral contactor (US2), and if the safety configuration specifies for this to switch in accordance with the motion enable, then the same also applies in Start-up mode. This means that if mo-

tion enable is present, the US2 voltage is switched on – even in Start-up mode.

NOTICE
The maximum number of switching cycles of the peripheral contactors is 175 per day.

Hazards

Possible hazards and risks involved in using Start-up mode:

- A person walks into the manipulator's danger zone.
- In a hazardous situation, a disabled external EMERGENCY STOP device is actuated and the manipulator is not shut down.

Additional measures for avoiding risks in Start-up mode:

- Cover disabled EMERGENCY STOP devices or attach a warning sign indicating that the EMERGENCY STOP device is out of operation.
- If there is no safety fence, other measures must be taken to prevent persons from entering the manipulator's danger zone, e.g. use of warning tape.

Use

Intended use of Start-up mode:

- Start-up in T1 mode when the external safeguards have not yet been installed or put into operation. The danger zone must be delimited at least by means of warning tape.
- Fault localization (periphery fault).
- Use of Start-up mode must be minimized as much as possible.

WARNING
Use of Start-up mode disables all external safeguards. The service personnel are responsible for ensuring that there is no-one in or near the danger zone of the manipulator as long as the safeguards are disabled. Failure to observe this precaution may result in death, injuries or damage to property.

Misuse

Any use or application deviating from the intended use is deemed to be misuse and is not allowed. KUKA Deutschland GmbH is not liable for any damage resulting from such misuse. The risk lies entirely with the user.

3.8.5 Manual mode

General

Manual mode is the mode for setup work. Setup work is all the tasks that have to be carried out on the industrial robot to enable automatic operation. Setup work includes:

- Jog mode
- Teaching
- Programming
- Program verification

The following must be taken into consideration in manual mode:

- New or modified programs must always be tested first in Manual Reduced Velocity mode (T1).
- The manipulator, tooling or external axes (optional) must never touch or project beyond the safety fence.
- Workpieces, tooling and other objects must not become jammed as a result of the industrial robot motion, nor must they lead to short-circuits or be liable to fall off.
- All setup work must be carried out, where possible, from outside the safeguarded area.

Setup work in T1

If it is necessary to carry out setup work from inside the safeguarded area, the following must be taken into consideration in the operating mode **Manual Reduced Velocity (T1)**:

- If it can be avoided, there must be no other persons inside the safeguarded area.
- If it is necessary for there to be several persons inside the safeguarded area, the following must be observed:
  - Each person must have an enabling device.
  - All persons must have an unimpeded view of the industrial robot.
  - Eye-contact between all persons must be possible at all times.
- The operator must be so positioned that he can see into the danger area and get out of harm's way.
- Unexpected motions of the manipulator cannot be ruled out, e.g. in the event of a fault. For this reason, an appropriate clearance must be maintained between persons and the manipulator (including tool). Guide value: 50 cm.

  The minimum clearance may vary depending on local circumstances, the motion program and other factors. The minimum clearance that is to apply for the specific application must be decided by the user on the basis of a risk assessment.

Setup work in T2

If it is necessary to carry out setup work from inside the safeguarded area, the following must be taken into consideration in the operating mode **Manual High Velocity (T2)**:

- This mode may only be used if the application requires a test at a velocity higher than that possible in T1 mode.
- Teaching and programming are not permissible in this operating mode.
- Before commencing the test, the operator must ensure that the enabling devices are operational.
- The operator must be positioned outside the danger zone.
- There must be no other persons inside the safeguarded area. It is the responsibility of the operator to ensure this.

3.8.6 Simulation

Simulation programs do not correspond exactly to reality. Robot programs created in simulation programs must be tested in the system in **Manual Reduced Velocity mode (T1)**. It may be necessary to modify the program.

3.8.7 Automatic mode

Automatic mode is only permissible in compliance with the following safety measures:

- All safety equipment and safeguards are present and operational.
- There are no persons in the system.
- The defined working procedures are adhered to.

If the manipulator or an external axis (optional) comes to a standstill for no apparent reason, the danger zone must not be entered until an EMERGENCY STOP has been triggered.

3.8.8 Maintenance and repair

After maintenance and repair work, checks must be carried out to ensure the required safety level. The valid national or regional work safety regulations must be observed for this check. The correct functioning of all safety functions must also be tested.

The purpose of maintenance and repair work is to ensure that the system is kept operational or, in the event of a fault, to return the system to an operational state. Repair work includes troubleshooting in addition to the actual repair itself.

The following safety measures must be carried out when working on the industrial robot:

- Carry out work outside the danger zone. If work inside the danger zone is necessary, the user must define additional safety measures to ensure the safe protection of personnel.
- Switch off the industrial robot and secure it (e.g. with a padlock) to prevent it from being switched on again. If it is necessary to carry out work with the robot controller switched on, the user must define additional safety measures to ensure the safe protection of personnel.
- If it is necessary to carry out work with the robot controller switched on, this may only be done in operating mode T1.
- Label the system with a sign indicating that work is in progress. This sign must remain in place, even during temporary interruptions to the work.
- The EMERGENCY STOP devices must remain active. If safety functions or safeguards are deactivated during maintenance or repair work, they must be reactivated immediately after the work is completed.

DANGER

Before work is commenced on live parts of the robot system, the main switch must be turned off and secured against being switched on again. The system must then be checked to ensure that it is deenergized.
It is not sufficient, before commencing work on live parts, to execute an EMERGENCY STOP or a safety stop, or to switch off the drives, as this does not disconnect the robot system from the mains power supply. Parts remain energized. Death or severe injuries may result.

Faulty components must be replaced using new components with the same article numbers or equivalent components approved by KUKA Deutschland GmbH for this purpose.

Cleaning and preventive maintenance work is to be carried out in accordance with the operating instructions.

Robot controller

Even when the robot controller is switched off, parts connected to peripheral devices may still carry voltage. The external power sources must therefore be switched off if work is to be carried out on the robot controller.

The ESD regulations must be adhered to when working on components in the robot controller.

Voltages in excess of 50 V (up to 780 V) can be present in various components for several minutes after the robot controller has been switched off! To prevent life-threatening injuries, no work may be carried out on the industrial robot in this time.

Water and dust must be prevented from entering the robot controller.

Counterbalancing system

Some robot variants are equipped with a hydropneumatic, spring or gas cylinder counterbalancing system.

The hydropneumatic and gas cylinder counterbalancing systems are pressure equipment and, as such, are subject to obligatory equipment monitoring and the provisions of the Pressure Equipment Directive.

The user must comply with the applicable national laws, regulations and standards pertaining to pressure equipment.

Inspection intervals in Germany in accordance with Industrial Safety Order, Sections 14 and 15. Inspection by the user before commissioning at the installation site.

The following safety measures must be carried out when working on the counterbalancing system:

- The manipulator assemblies supported by the counterbalancing systems must be secured.
- Work on the counterbalancing systems must only be carried out by qualified personnel.

Hazardous substances

The following safety measures must be carried out when handling hazardous substances:

- Avoid prolonged and repeated intensive contact with the skin.
- Avoid breathing in oil spray or vapors.
- Clean skin and apply skin cream.

To ensure safe use of our products, we recommend regularly requesting up-to-date safety data sheets for hazardous substances.

3.8.9 Decommissioning, storage and disposal

The industrial robot must be decommissioned, stored and disposed of in accordance with the applicable national laws, regulations and standards.

3.8.10 Safety measures for “single point of control”

Overview

If certain components in the industrial robot are operated, safety measures must be taken to ensure complete implementation of the principle of “single point of control” (SPOC).

The relevant components are:

- Submit interpreter
- PLC
- OPC server
- Remote control tools
- Tools for configuration of bus systems with online functionality
- KUKA.RobotSensorInterface

The implementation of additional safety measures may be required. This must be clarified for each specific application; this is the responsibility of the system integrator, programmer or user of the system.

Since only the system integrator knows the safe states of actuators in the periphery of the robot controller, it is his task to set these actuators to a safe state, e.g. in the event of an EMERGENCY STOP.

T1, T2

In modes T1 and T2, the components referred to above may only access the industrial robot if the following signals have the following states:

Signal	State required for SPOC
$USER_SAF	TRUE
$SPOC_MOTION_ENABLE	TRUE

Submit interpreter, PLC

If motions, (e.g. drives or grippers) are controlled with the submit interpreter or the PLC via the I/O system, and if they are not safeguarded by other means, then this control will take effect even in T1 and T2 modes or while an EMERGENCY STOP is active.

If variables that affect the robot motion (e.g. override) are modified with the submit interpreter or the PLC, this takes effect even in T1 and T2 modes or while an EMERGENCY STOP is active.

Safety measures:

- In T1 and T2, the system variable $OV_PRO must not be written to by the submit interpreter or the PLC.
- Do not modify safety-relevant signals and variables (e.g. operating mode, EMERGENCY STOP, safety gate contact) via the submit interpreter or PLC.

  If modifications are nonetheless required, all safety-relevant signals and variables must be linked in such a way that they cannot be set to a dangerous state by the submit interpreter or PLC. This is the responsibility of the system integrator.

OPC server, remote control tools

These components can be used with write access to modify programs, outputs or other parameters of the robot controller, without this being noticed by any persons located inside the system.

Safety measure:

If these components are used, outputs that could cause a hazard must be determined in a risk assessment. These outputs must be designed in such a way that they cannot be set without being enabled. This can be done using an external enabling device, for example.

Tools for configuration of bus systems

If these components have an online functionality, they can be used with write access to modify programs, outputs or other parameters of the robot controller, without this being noticed by any persons located inside the system.

- WorkVisual from KUKA
- Tools from other manufacturers

Safety measure:

In the test modes, programs, outputs or other parameters of the robot controller must not be modified using these components.

3.9 Applied norms and regulations

Name/Edition	Definition
2006/42/EC:2006	**Machinery Directive:** Directive 2006/42/EC of the European Parliament and of the Council of 17 May 2006 on machinery, and amending Directive 95/16/EC (recast)
2014/30/EU:2014	**EMC Directive:** Directive 2014/30/EC of the European Parliament and of the Council dated 26 February 2014 on the approximation of the laws of the Member States concerning electromagnetic compatibility
2014/68/EU:2014	**Pressure Equipment Directive**: Directive 2014/68/EU of the European Parliament and of the Council dated 15 May 2014 on the approximation of the laws of the Member States concerning pressure equipment (Only applicable for robots with hydropneumatic counterbalancing system.)
EN ISO 13850:2015	**Safety of machinery:** Emergency stop - Principles for design
EN ISO 13849-1:2015	**Safety of machinery:** Safety-related parts of control systems - Part 1: General principles of design
EN ISO 13849-2:2012	**Safety of machinery:** Safety-related parts of control systems - Part 2: Validation
EN ISO 12100:2010	**Safety of machinery:** General principles of design, risk assessment and risk reduction
EN ISO 10218-1:2011	**Industrial robots – Safety requirements:** Part 1: Robots **Note:** Content equivalent to **ANSI/RIA R.15.06-2012, Part 1**
EN 614-1:2006+A1:2009	**Safety of machinery:** Ergonomic design principles - Part 1: Terms and general principles

EN 61000-6-2:2005	**Electromagnetic compatibility (EMC):** Part 6-2: Generic standards; Immunity for industrial environments
EN 61000-6-4:2007 + A1:2011	**Electromagnetic compatibility (EMC):** Part 6-4: Generic standards; Emission standard for industrial environments
EN 60204-1:2006/ A1:2009	**Safety of machinery:** Electrical equipment of machines - Part 1: General requirements

4 Operation

4.1 KUKA smartPAD teach pendant

The smartPAD is the teach pendant for the industrial robot. The smartPAD has all the operator control and display functions required for operating and programming the industrial robot.

Two models exist:

- **smartPAD**
- **smartPAD-2**

In this documentation, the designation “KUKA smartPAD” or “smartPAD” refers to both models unless an explicit distinction is made.

4.1.1 smartPAD

4.1.1.1 Front of smartPAD

The smartPAD has a touch screen: the smartHMI can be operated with a finger or stylus. An external mouse or external keyboard is not necessary.

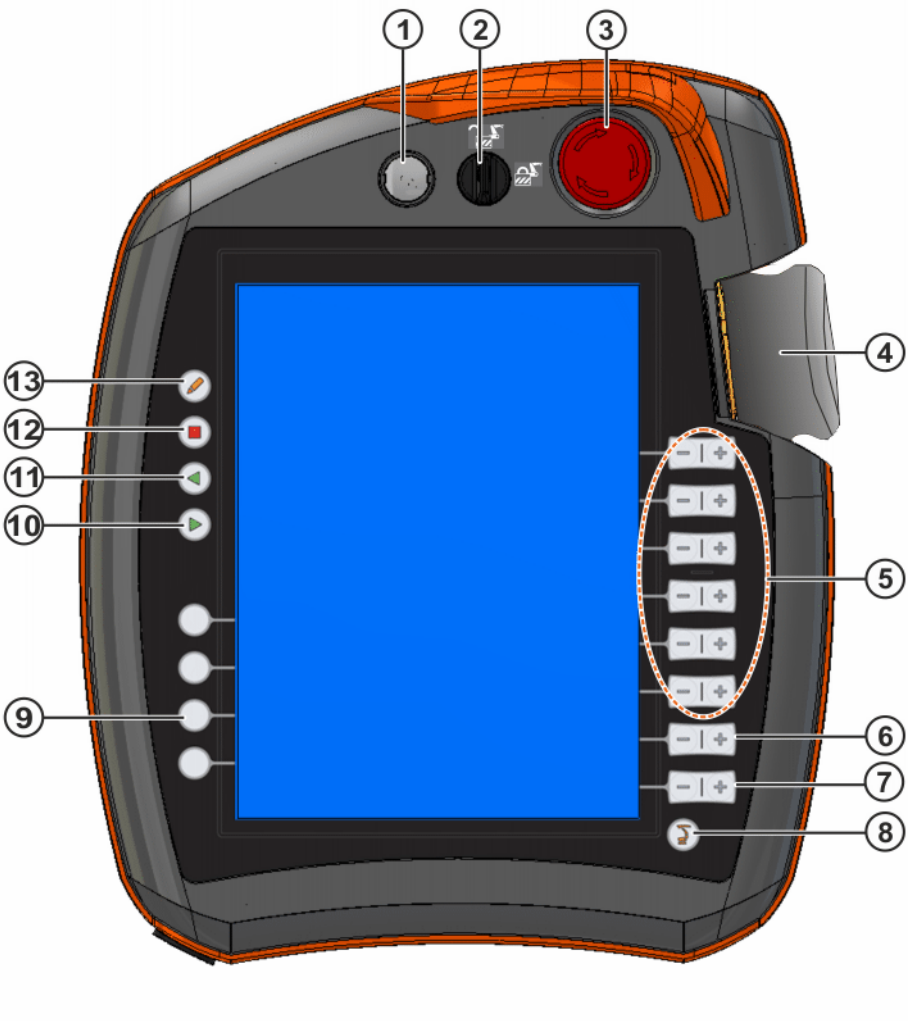


Fig. 4-1: Front of smartPAD

Item	Description
1	Button for disconnecting the smartPAD (>>> *4.1.3 "Disconnecting and connecting the smartPAD" Page 59*)
2	Mode selector switch: the switch may be one of the following variants: • With key It is only possible to change operating mode if the key is inserted. • Without key The mode selector switch is used to call the connection manager. The connection manager is used to change the operating mode. (>>> *4.14 "Changing operating mode" Page 78*)
3	EMERGENCY STOP device: stops the robot in hazardous situations. The EMERGENCY STOP device locks itself in place when it is pressed.
4	Space Mouse: for moving the robot manually
5	Jog keys: for moving the robot manually
6	Key for setting the program override
7	Key for setting the jog override
8	Main menu key: shows the menu items on the smartHMI. It can also be used for creating screenshots.
9	Status buttons: the status keys are used primarily for setting parameters in option packages. Their exact function depends on the option packages installed.
10	Start key: the Start key is used to start a program.
11	Start backwards key: the Start backwards key is used to start a program backwards. The program is executed step by step.
12	STOP key: the STOP key is used to stop a program that is running.
13	Keyboard key: Displays the keyboard. It is generally not necessary to press this key to display the keyboard, as the smartHMI detects when keyboard input is required and displays the keyboard automatically. (>>> *4.2.1 "Keypad" Page 62*)

4.1.1.2 Rear of smartPAD

Fig. 4-2: Rear of smartPAD

Item	Description
1	Enabling switch The enabling switches have 3 positions: • Not pressed • Center position • Fully pressed (panic position) In the test modes, the manipulator can only be moved if at least one of the enabling switches is held in the center position. In the Automatic and Automatic External modes, the enabling switches have no function.
2	Start key (green): the Start key is used to start a program.
3	Enabling switch
4	USB connection: it is used, for example, for archiving and restoring data. Only for FAT32-formatted USB sticks.
5	Enabling switch
6	Identification plate

4.1.2 smartPAD-2

4.1.2.1 Front of smartPAD-2

The smartPAD-2 has a capacitive touch screen: the smartHMI can be operated with a finger or capacitive stylus. An external mouse or external keyboard is not necessary.

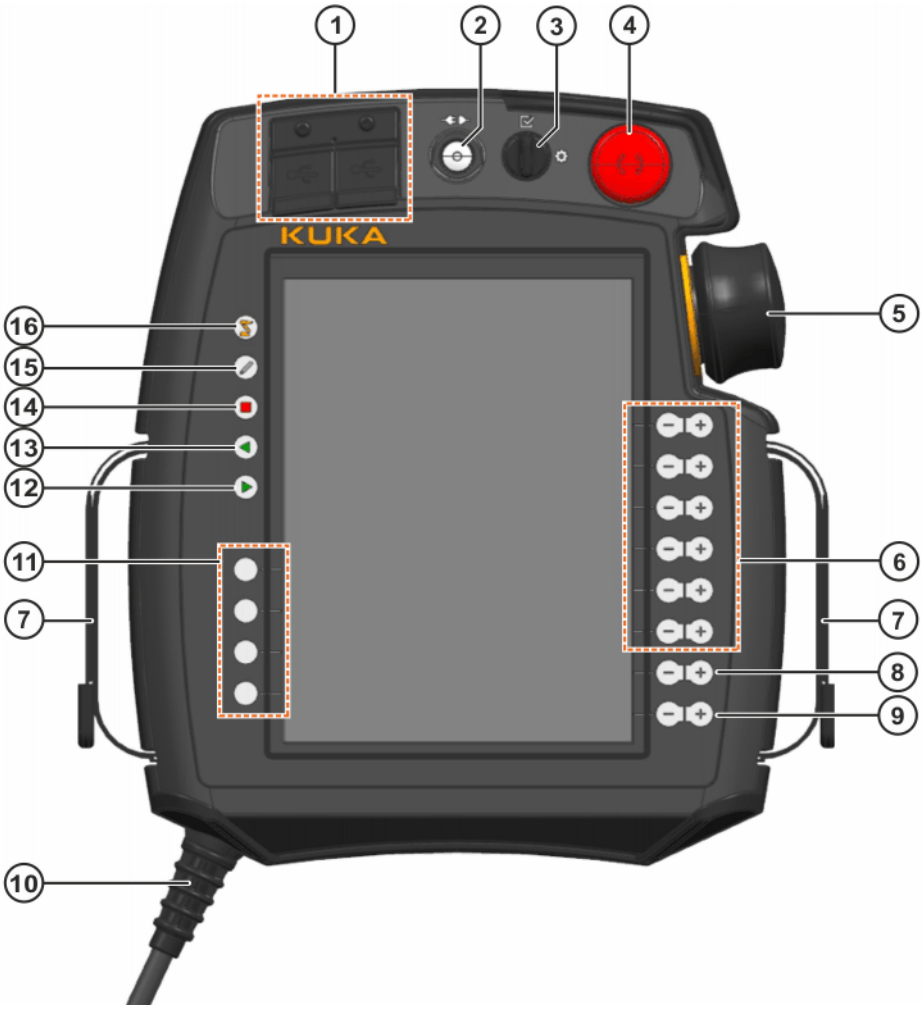


Fig. 4-3: Front of smartPAD-2

Item	Description
1	2 USB 2.0 interfaces with cover The USB connection is used for archiving data, for example. For NTFS and FAT32-formatted USB sticks.
2	Button for disconnecting the smartPAD (>>> *4.1.3 "Disconnecting and connecting the smartPAD" Page 59*)
3	Mode selector switch: the switch may be one of the following variants: • With key It is only possible to change operating mode if the key is inserted. • Without key The mode selector switch is used to call the connection manager. The connection manager is used to change the operating mode. (>>> *4.14 "Changing operating mode" Page 78*)

Item	Description
4	EMERGENCY STOP device: stops the robot in hazardous situations. The EMERGENCY STOP device locks itself in place when it is pressed.
5	Space Mouse (6D mouse): for moving the robot manually
6	Jog keys: for moving the robot manually
7	Hand straps with Velcro fastener: when the hand straps are not in use, they can be pulled in completely.
8	Key for setting the program override
9	Key for setting the jog override
10	Connecting cable
11	Status buttons: the status keys are used primarily for setting parameters in option packages. Their exact function depends on the option packages installed.
12	Start key: the Start key is used to start a program.
13	Start backwards key: the Start backwards key is used to start a program backwards. The program is executed step by step.
14	STOP key: the STOP key is used to stop a program that is running.
15	Keyboard key: Displays the keyboard. It is generally not necessary to press this key to display the keyboard, as the smartHMI detects when keyboard input is required and displays the keyboard automatically. (>>> *4.2.1 "Keypad" Page 62*)
16	Main menu key: the main menu key shows and hides the main menu on the smartHMI. It can also be used for creating screenshots.

4.1.2.2 Rear of smartPAD-2

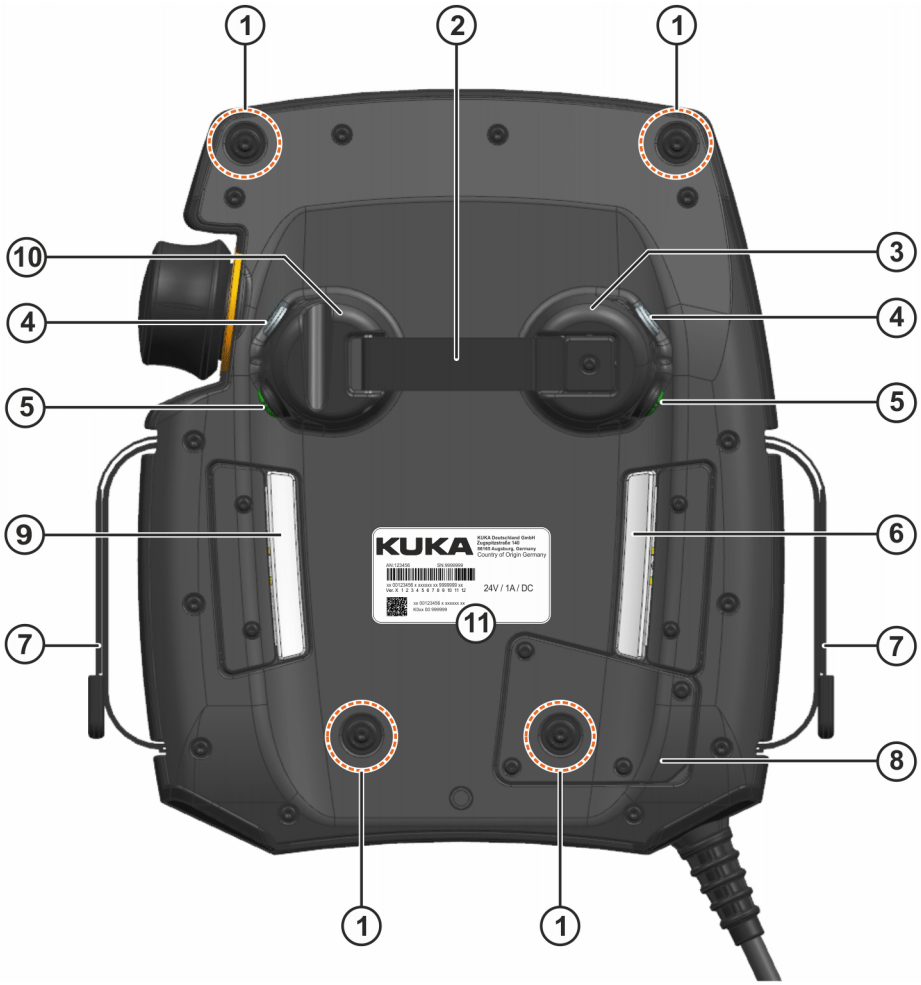


Fig. 4-4: Rear of smartPAD-2

Item	Description
1	Press studs for fastening the (optional) carrying strap
2	Strap, dome
3	Left-hand dome: holding the smartPAD with the right hand
4	Enabling switch The enabling switches have 3 positions: • Not pressed • Center position • Fully pressed (panic position) In the test modes, the manipulator can only be moved if at least one of the enabling switches is held in the center position. In the Automatic and Automatic External modes, the enabling switches have no function.
5	Start key (green): the Start key is used to start a program.
6	Enabling switch
7	Hand straps with Velcro fastener: when the hand straps are not in use, they can be pulled in completely.
8	Cover (connection cable cover)

Item	Description
9	Enabling switch
10	Right-hand dome: holding the smartPAD with the left hand
11	Identification plate

4.1.3 Disconnecting and connecting the smartPAD

Description

The information is applicable for both the smartPAD and the smartPAD-2.

WARNING

If the smartPAD is disconnected, the system can no longer be switched off by means of the EMERGENCY STOP device on the smartPAD. For this reason, an external EMERGENCY STOP must be connected to the robot controller.
The user is responsible for ensuring that the smartPAD is immediately removed from the system when it has been disconnected. The smartPAD must be stored out of sight and reach of personnel working on the robot. This prevents operational and non-operational EMERGENCY STOP devices from becoming interchanged.
Failure to observe these precautions may result in death, injuries or damage to property.

The “smartPAD” and “smartPAD-2” models are compatible with one another, i.e. if one model has been disconnected, then the other model can then be plugged in.

It must be ensured that the Space Mouse (6D mouse) is not deflected when connecting the smartPAD to a controller that is runnning or when switching on the controller. Otherwise the mouse will not function correctly.

Procedure

Disconnection:

The smartPAD can also be disconnected while the robot controller is running.

1. Press the disconnect button on the smartPAD.

   A message and a counter are displayed on the smartHMI. The counter runs for 25 s. During this time, the smartPAD can be disconnected from the robot controller.

   If the counter expires without the smartPAD having been disconnected, this has no effect. The disconnect button can be pressed again at any time to display the counter again.
2. Disconnect the smartPAD from the robot controller.

If the smartPAD is disconnected without the counter running, this triggers an EMERGENCY STOP. The EMERGENCY STOP can only be canceled by plugging the smartPAD back in.

Connection:

A smartPAD can be connected at any time.

1. Connect the smartPAD to the robot controller.

- The EMERGENCY STOP and enabling switches are operational again 30 s after connection.
- The smartHMI is automatically displayed again. (This may take longer than 30 s.)
- The connected smartPAD assumes the current operating mode of the robot controller.

The current operating mode is not, in all cases, the same as that before the smartPAD was disconnected: if the robot controller is part of a RoboTeam, the operating mode may have been changed after disconnection, e.g. by the master.

2. Check the functions. The following checks must be performed:
   - Function test of EMERGENCY STOP
   - Function test for the enabling switches
     (>>> *"Function test" Page 34*)
   - Check whether the smartHMI is displayed again. (This may take longer than 30 s.)

WARNING

Disconnect a non-operational smartPAD and remove it from the system! If a non-operational smartPAD remains connected, there is the danger that a user will attempt to activate a non-operational EMERGENCY STOP. Death, injuries or damage to property may result.

4.2 KUKA smartHMI user interface

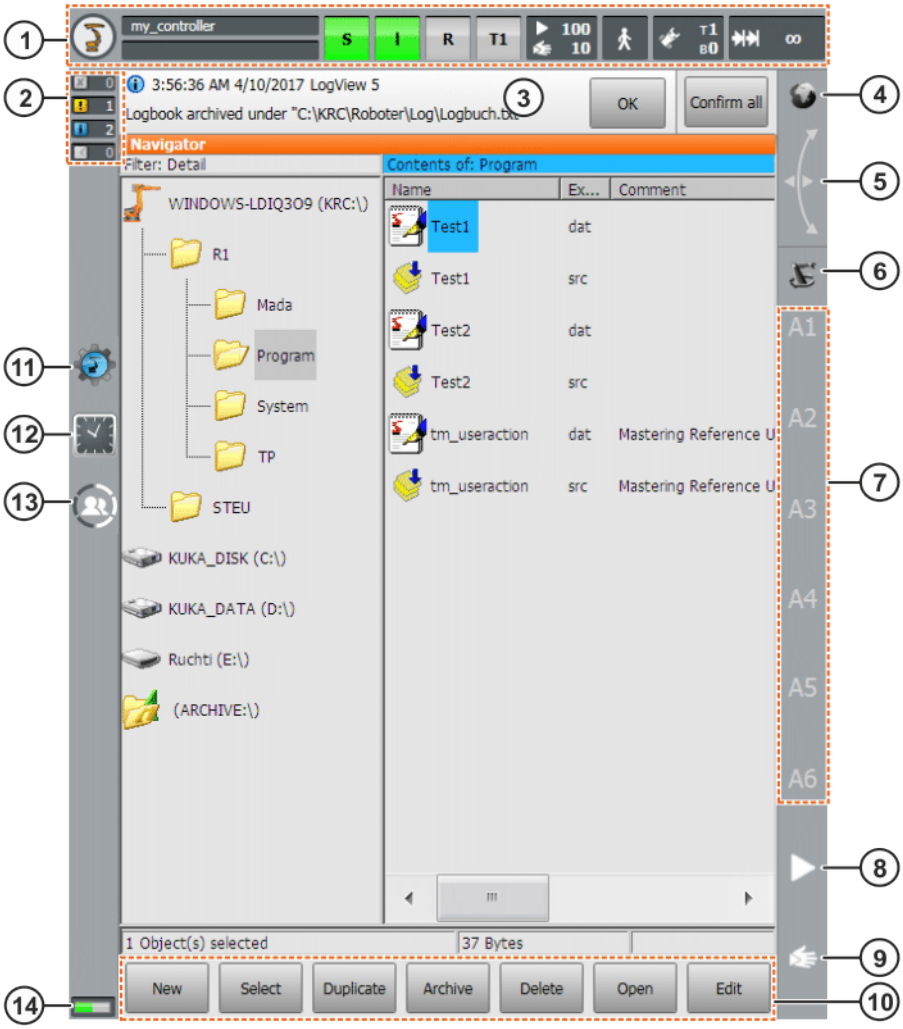


Fig. 4-5: KUKA smartHMI user interface

Item	Description
1	Status bar (>>> *4.2.2 "Status bar" Page 63*)
2	Message counter The message counter indicates how many messages of each message type are active. Touching the message counter enlarges the display.
3	Message window By default, only the last message is displayed. Touching the message window expands it so that all active messages are displayed. An acknowledgeable message can be acknowledged with **OK**. All acknowledgeable messages can be acknowledged at once with **Confirm all**.
4	**Space Mouse** status indicator This indicator shows the current coordinate system for jogging with the Space Mouse. Touching the indicator displays all coordinate systems, allowing a different one to be selected. Required user rights: function group **General jog settings**

Item	Description
5	**Space Mouse alignment** indicator Touching this indicator opens a window in which the current alignment of the Space Mouse is indicated and can be changed. (>>> *4.16.7 "Defining the alignment of the Space Mouse" Page 90*)
6	**Jog keys** status indicator This indicator shows the current coordinate system for jogging with the jog keys. Touching the indicator displays all coordinate systems, allowing a different one to be selected. Required user rights: function group **General jog settings**
7	Jog key labels If axis-specific jogging is selected, the axis numbers are displayed here (A1, A2, etc.). If Cartesian jogging is selected, the coordinate system axes are displayed here (X, Y, Z, A, B, C). Touching the label indicates which kinematics group is selected.
8	Program override (>>> *8.5 "Setting the program override" Page 331*)
9	Jog override (>>> *4.16.2 "Setting the jog override" Page 86*)
10	Button bar. The buttons change dynamically and always refer to the window that is currently active in the smartHMI. At the right-hand end is the **Edit** button. This can be used to call numerous commands relating to the Navigator.
11	WorkVisual icon Touching the icon takes you to the **Project management** window.
12	Clock The clock displays the system time. Touching the clock displays the system time in digital format, together with the current date.
13	**User group** icon The number of white segments in the circle denotes which user group is currently selected. Touching the icon opens the user group selection window. (>>> *4.12 "Changing user group" Page 76*)
14	Life sign display If the display flashes in the following manner, this indicates that the smartHMI is active: The left-hand and right-hand lamps alternately light up green. The change is slow (approx. 3 s) and uniform.

4.2.1 Keypad

The smartPAD has a touch screen: the smartHMI can be operated with a finger or stylus.

There is a keypad on the smartHMI for entering letters and numbers. The smartHMI detects when the entry of letters or numbers is required and automatically displays the keypad.

The keypad only ever displays the characters that are required. If, for example, a box is edited in which only numbers can be entered, then only numbers are displayed and not letters.

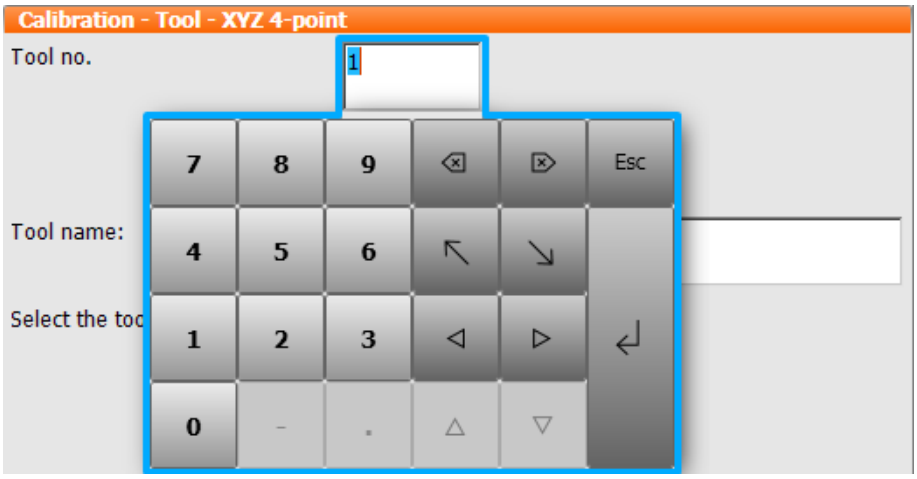


Fig. 4-6: Example keypad

4.2.2 Status bar

The status bar indicates the status of certain central settings of the industrial robot. In most cases, touching the display opens a window in which the settings can be modified.

Overview

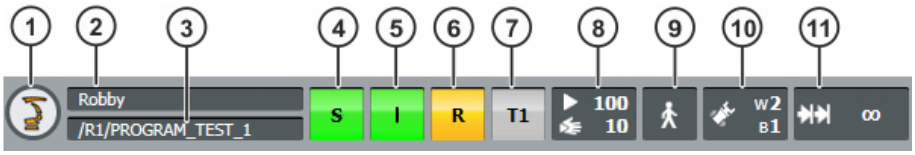


Fig. 4-7: KUKA smartHMI status bar

Item	Description
1	Main menu key. Shows the menu items on the smartHMI.
2	Name of the robot controller
3	If a program has been selected, the name is displayed here.
4	**Submit interpreter** status indicator (>>> *12.2 "Status indicator for the submit interpreter" Page 574*)
5	**Drives** status indicator. Touching the display opens a window in which the drives can be switched on or off. (>>> *4.2.3 "Drives status indicator and Motion conditions window" Page 64*)
6	**Robot interpreter** status indicator. Programs can be reset or canceled here. (>>> *8.6 "Robot interpreter status indicator" Page 332*)
7	Current operating mode
8	**Overrides** status indicator. Indicates the current program override and the current jog override.
9	**Program run mode** status indicator. Indicates the current program run mode.

Item	Description
10	**Cur. tool/base** status indicator. Indicates the current tool and base.
11	**Incremental jogging** status indicator

4.2.3 Drives status indicator and Motion conditions window

Drives status indicator

The **Drives** status indicator can display the following statuses:

Statuses			

Meaning of the symbols and colors:

Symbol: **I**	The drives are ON. ($PERI_RDY == TRUE) • The intermediate circuit is fully charged.
Symbol: **O**	The drives are OFF. ($PERI_RDY == FALSE) • The intermediate circuit is not charged or incompletely charged.
Color: Green	$COULD_START_MOTION == TRUE • The enabling switch has been pressed (center position) or is not required. • And: There are no active messages preventing robot motion.
Color: Gray	$COULD_START_MOTION == FALSE • The enabling switch has not been pressed or fully pressed. • And/or: There are active messages preventing robot motion.

- Drives ON does not automatically mean that the KSPs switch to servo-control and supply the motors with current.
- Drives OFF does not automatically mean that the KSPs terminate the power supply to the motors.

Whether or not the KSPs supply the motors with current depends on whether the drives enable signal has been received from the safety controller.

“Motion conditions” window

Touching the **Drives** status indicator opens the **Motion conditions** window. The drives can be switched on or off here.

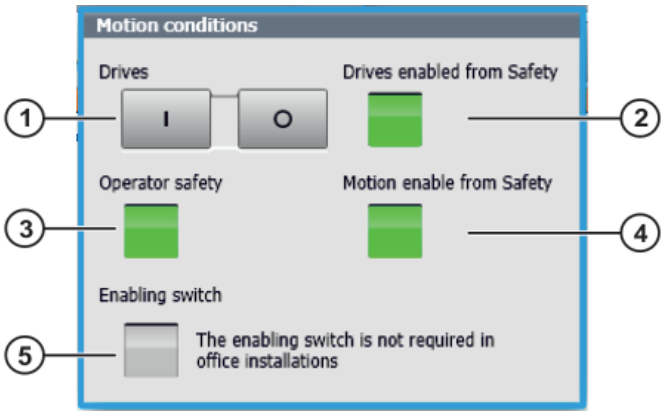


Fig. 4-8: “Motion conditions” window

Item	Description
1	**I**: Touch to switch on the drives. **O**: Touch to switch off the drives. Required user rights: Function group **Program selection and deselection**
2	**Green**: The drives enable signal has been received from the safety controller. **Gray**: The safety controller has triggered a safety stop 0 or terminated a safety stop 1. No drives enable signal present, i.e. the KSPs are not under servo-control and are not supplying the motors with current.
3	**Operator safety** signal **Green**: $USER_SAF == TRUE **Gray**: $USER_SAF == FALSE (>>> *"$USER_SAF == TRUE" Page 65*)
4	**Green**: The motion enable signal has been received from the safety controller. **Gray**: The safety controller has triggered a safety stop 1 or a safety stop 2. No motion enable present. **Note:** The status of **Motion enable from Safety** does not correlate with the status of $MOVE_ENABLE!
5	**Green**: The enabling switch is pressed (center position). **Gray**: The enabling switch has not been pressed or fully pressed, or is not required.

$USER_SAF == TRUE

The conditions under which $USER_SAF is TRUE depend on the controller variant and the operating mode:

Control	Operating mode	Condition
KR C4	T1, T2	• The enabling switch is pressed.
	AUT, AUT EXT	• The physical safeguard is closed.

Control	Operating mode	Condition
VKR C4	T1	• The enabling switch is pressed. • E2/E22 is closed.
	T2	• The enabling switch is pressed. • E2/E22 and E7 are closed
	EXT	• The physical safeguard is closed. • E2/E22 and E7 are open.

4.2.4 Minimizing KUKA smartHMI (displaying Windows interface)

Precondition

- User rights: Function group **Critical configurations**
- T1 or T2 mode

Procedure

1. In the main menu, select **Start-up** > **Service** > **Minimize HMI**.
   The smartHMI is minimized and the Windows interface is displayed.
2. To maximize the smartHMI again, touch the following icon in the task bar:

4.3 Switching on the robot controller and starting the KSS

Procedure

- Turn the main switch on the robot controller to ON.
  The operating system and the KSS start automatically.

If the KSS does not start automatically, e.g. because the Startup function has been disabled, execute the file StartKRC.exe in the directory C:\KRC.

If the robot controller is logged onto the network, the start may take longer.

> If the robot controller has been switched on with a kinematic system connected for the first time, a certain procedure must then be observed. (>>> *5.2 "Switching on the robot controller with the kinematic system for the first time" Page 125*)

4.4 Calling the main menu

Procedure

- Press the main menu key on the smartPAD. The **Main menu** window opens.
  The display is always the same as that which was in the window before it was last closed.

Description

Properties of the **Main menu** window:

- The main menu is displayed in the left-hand column.
- Touching a menu item that contains an arrow opens the corresponding submenu (e.g. **Display**).

  Depending on how many submenu levels are open, the **Main menu** column may no longer be visible, with only the submenus remaining visible.
- The Home key in the top right-hand corner closes all open submenus.
- The arrow key to the right of the Home key closes the most recently opened submenu.
- The most recently selected menu items are displayed under **Quick access** (maximum 4).

  The entries can be pinned or unpinned by touching the pin icon. A pinned entry is retained under **Quick access** and is not suppressed by the next submenu item called.
- The white cross on the left-hand side closes the window.

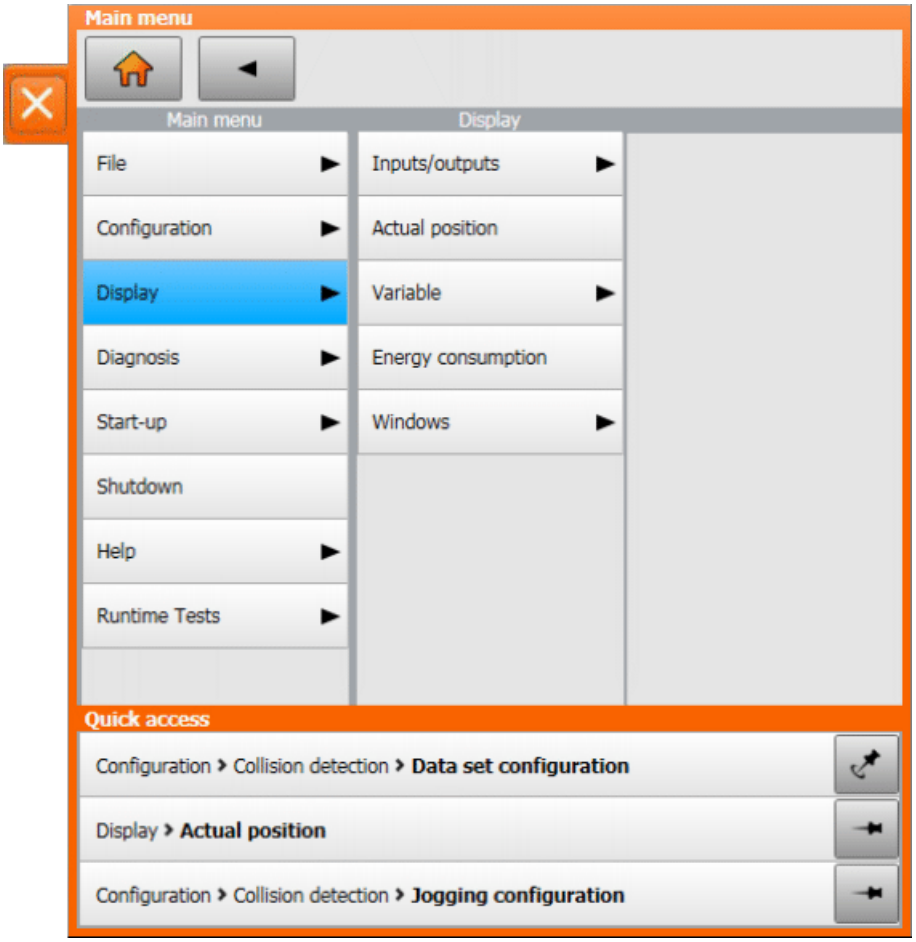


Fig. 4-9: Example: The Display submenu is open.

4.5 Defining the start type for KSS

Description

You can choose whether to boot the robot controller by default with a cold start or with Hibernate.

In the following situations, the robot controller always performs an initial cold start, irrespective of what start type has been defined:
- Following installation or update of the KSS
- When the robot controller has detected an error while shutting down

Precondition

- User rights: Function group **Critical configurations**

Procedure

1. In the main menu, select **Shutdown**. A window opens.
2. Select the start type: **Cold start** or **Hibernate**.
   (>>> *"Start types" Page 71*)
3. Close the window. The selected start type is applied.

It is possible to select settings which deviate from the standard start type and are valid only for the next start.
(>>> *4.6 "Shutting down or rebooting the robot controller" Page 68*)

4.6 Shutting down or rebooting the robot controller

Description

The KSS starts in whatever operating mode was most recently selected. Exceptions:

- If the most recent operating mode was T2, the mode after starting is T1.
- After an initial cold start, the operating mode is T1.

NOTICE

If the option **Reboot control PC** is selected, the main switch on the robot controller must not be pressed until the reboot has been completed. System files may otherwise be destroyed.
If this option was not selected, the main switch can be pressed once the controller has shut down.

Precondition

- User rights: Function group **General configuration**
  Or higher, depending on the desired options
- T1 or T2 mode

Procedure

1. Select the menu item **Shutdown** in the main menu.
2. Select the desired options.
3. Press **Shut down control PC** or **Reboot control PC**.
4. Confirm the request for confirmation with **Yes**. The System Software is terminated and restarted in accordance with the selected option.

After the restart, the following message is displayed:

- *Cold start of controller*
- Or, if **Reload files** has been selected: *Initial cold start of controller*

“Shutdown” window

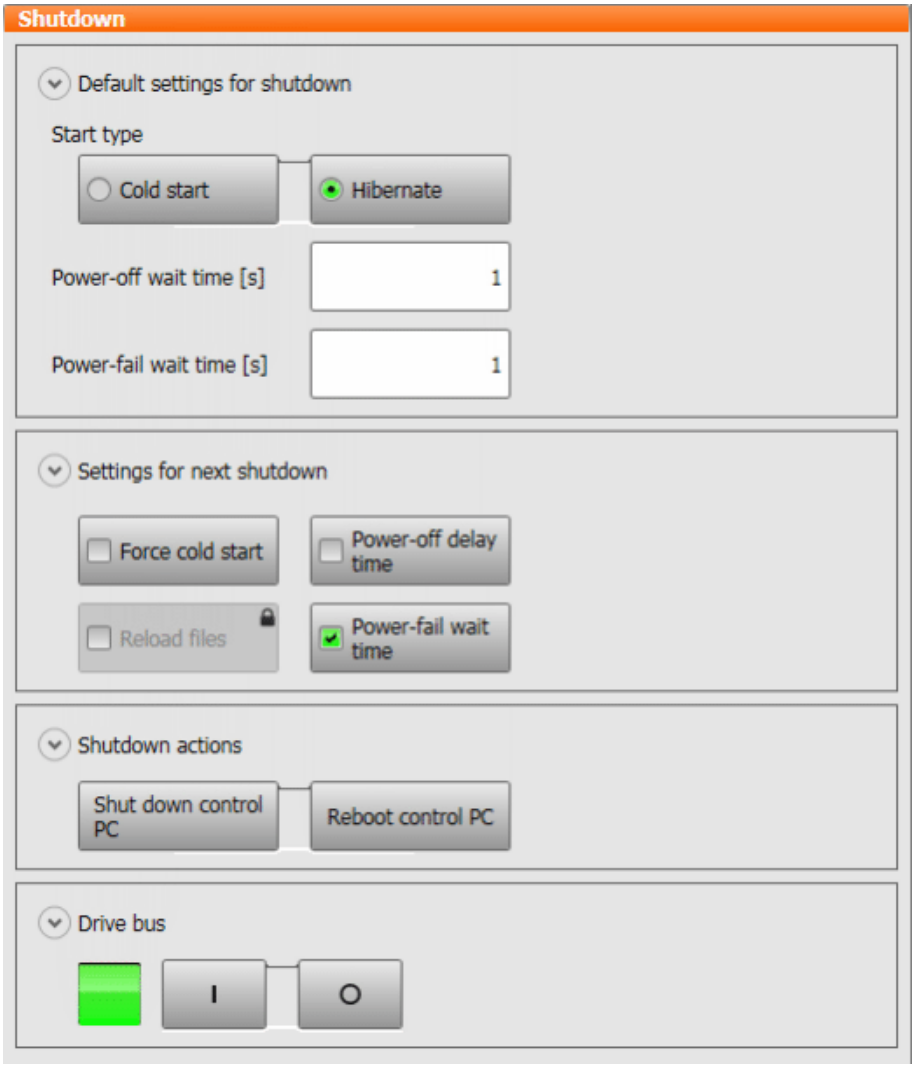


Fig. 4-10: “Shutdown” window

Option	Description
Default settings for shutdown Required user rights: Function group **Critical configurations**	
Cold start **Hibernate**	**Hibernate** is the standard start type. (>>> *"Start types" Page 71*)
Power-off delay time	If the robot controller is switched off at the main switch, it is only shut down after the delay time defined here. During the delay time, the robot controller is powered by its battery. • 0 … 1,000 s **Note:** The power-off delay time has no function for the controller variant “KR C4 compact”. The power failure delay time applies here, even when switching off via the main switch.

Option	Description
Power-fail wait time	In the case of a power failure, the robot comes to a standstill. However, the robot controller does not shut down immediately but rather only after the power failure delay time. In other words, brief power failures are overridden through this delay time. The error messages must then only be acknowledged and the program can be resumed. During the delay time, the robot controller is powered by its battery. • 0 … 1,000 s If the power failure lasts longer than the power failure delay time and the robot controller shuts down, then the standard start type defined in the **Shutdown** window applies for the restart.
Settings for next shutdown Required user rights: Function group **Critical configurations**, except **Force cold start**.	
Force cold start	Active: The next start is a cold start. Only available if **Hibernate** has been selected. Required user rights: Function group **General configuration**
Reload files	Active: The next start is an initial cold start. This option must be selected in the following cases: • If XML files have been changed directly, i.e. the user has opened the file and modified it. (Any other changes to XML files, e.g. if the robot controller modifies them in the background, are irrelevant.) • If hardware components are to be exchanged after shutdown. Only available if **Cold start** or **Force cold start** has been selected. Depending on the hardware, the initial cold start takes approx. 30 to 150 seconds longer than a normal cold start.
Power-off delay time **Power-fail wait time**	Active: The delay time is adhered to the next time the system is shut down. Inactive: The delay time is ignored the next time the system is shut down.
Shutdown actions These options are only available in T1 and T2. Required user rights: Function group **General configuration**	
Shut down control PC	The robot controller shuts down.
Reboot control PC	The robot controller shuts down and then reboots with a cold start.
Drive bus Required user rights: Function group **General configuration**	
OFF / **ON**	The drive bus can be switched off or on. **Drive bus status** indicator: • Green: drive bus is on. • Red: drive bus is off. • Gray: status of the drive bus is unknown.

Start types

Start type	Description
Cold start	After a cold start the robot controller displays the Navigator. No program is selected. The robot controller is reinitialized, e.g. all user outputs are set to FALSE. **Note:** If XML files have been changed directly, i.e. the user has opened the file and modified it, these changes are taken into consideration in the case of a cold start with **Reload files**. This cold start is called an “initial cold start”. In the case of a cold start without **Reload files**, these changes are not taken into consideration.
Hibernate	After a start with Hibernate, the previously selected robot program can be resumed. The state of the kernel system: programs, block pointer, variable contents and outputs, is completely restored. Additionally, all programs that were open parallel to the robot controller are reopened and have the same state that they had before the system was shut down. The last state of Windows is also restored.

4.7 Switching drives on/off

Precondition

- User rights: Function group **Program selection and deselection**

Procedure

1. In the status bar, touch the **Drives** status indicator. The **Motion conditions** window opens.
   (>>> *4.2.3 "Drives status indicator and Motion conditions window" Page 64*)
2. Switch the drives on or off.

4.8 Switching the robot controller off

Description

When the system is switched off, the robot stops and the robot controller shuts down. The robot controller automatically backs up data.

If a power-off delay time is configured, the robot controller shuts down only after this time has passed. In other words, brief power-downs are overriden through this delay time. The error messages must then only be acknowledged and the program can then be resumed.

Exception: the power-off delay time has no function for the controller variant “KR C4 compact”. The power failure delay time applies here, even when switching off via the main switch.

During the delay time, the robot controller is powered by its battery.

Procedure

- Turn the main switch on the robot controller to OFF.

NOTICE
The main switch must not be operated if the robot controller has been exited with the option **Reboot control PC** and the reboot has not yet been completed. System files may otherwise be destroyed.

4.9 Setting the user interface language

Precondition

- User rights: Function group **General configuration**

Procedure

1. In the main menu, select **Configuration** > **Miscellaneous** > **Language**.
2. Select the desired language. Confirm with **OK**.

Description

The following languages are available:

Chinese (simplified)	Polish
Danish	Portuguese
German	Romanian
English	Russian
Finnish	Swedish
French	Slovak
Greek	Slovenian
Italian	Spanish
Japanese	Czech
Korean	Turkish
Dutch	Hungarian

4.10 Creating a screenshot on the smartPAD

Procedure

- Press the main menu key (= key with robot icon) twice in quick succession.

The screenshot is saved in the directory C:/KUKA/Screenshot.

4.11 Online documentation and help for messages

4.11.1 Calling online documentation

Description

The documentation of the System Software can be displayed on the robot controller. Certain technology packages also have documentation that can be displayed on the robot controller.

Procedure

1. In the main menu, select **Help** > **Documentation**. Then select either **System Software** or the menu item for the technology package.

   The **KUKA Embedded Information Service** window is opened. The table of contents of the documentation is displayed.
2. Touch a chapter. The topics it contains are displayed.
3. Touch a topic. The description is displayed.

Example

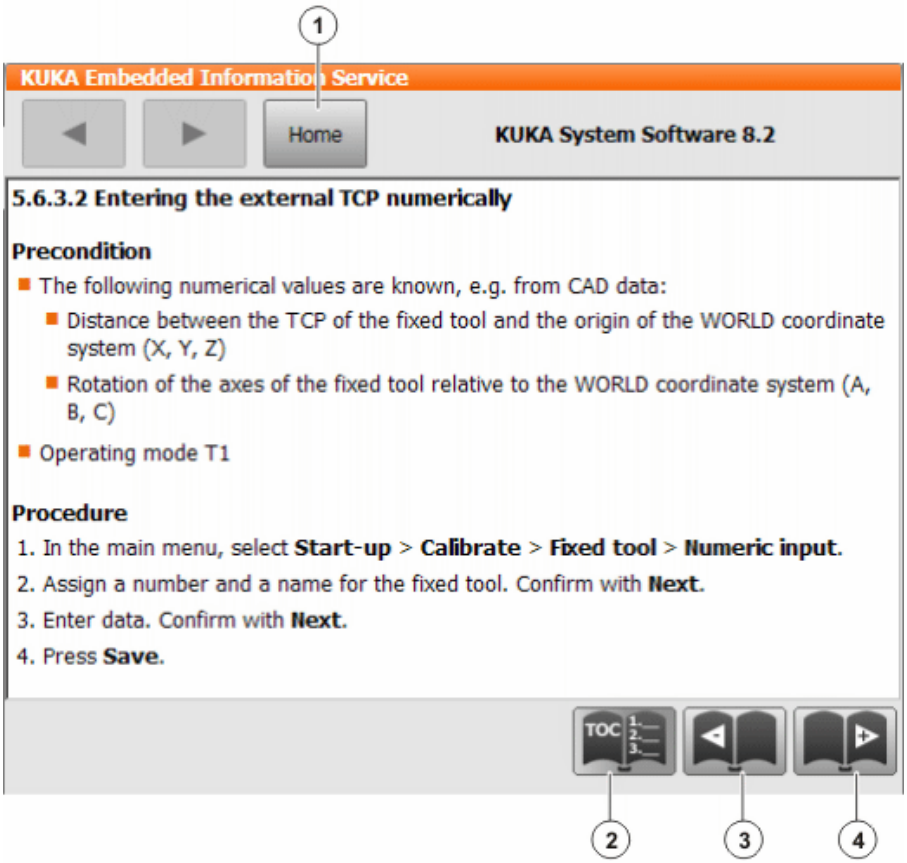


Fig. 4-11: Online documentation – Example from the KUKA System Software

Item	Description
1, 2	Displays the table of contents.
3	Displays the previous topic in the table of contents.
4	Displays the next topic.

4.11.2 Calling help for the messages

Description

The help for a message can be called in the following ways:

- Call the help for a specific message that is currently displayed in the message window.
- Display an overview of the possible messages and call the help for a message there.

Procedure

Calling the help for a message in the message window

Most messages contain a button with a question mark. Help is available for these messages.

1. Touch the question mark. The **KUKA Embedded Information Service – Message page** window is opened.

   The window contains a variety of information about the message. (>>> *Fig. 4-12*)

2. The window often also contains information about the causes of the message and the corresponding solutions. Details can be displayed:
   a. Touch the magnifying glass icon next to the cause. The detail page is opened. (>>> *Fig. 4-13*)
   b. Open the descriptions of the cause and solution.
   c. If the message has several possible causes: the magnifying glass icons with arrows can be used to jump to the previous or next detail page.

Procedure

Display an overview of the messages and call the help for a message.

1. In the main menu, select **Help** > **Messages**. Then select either **System Software** or the menu item for the technology package.

   The **KUKA Embedded Information Service** – **Index page** window is opened. The messages are sorted by module (“module” refers here to a subsection of the software).
2. Touch an entry. The messages of this module are displayed.
3. Touch a message. The message page is displayed.

   The window contains a variety of information about the message. (>>> *Fig. 4-12*)
4. The window often also contains information about the causes of the message and the corresponding solutions. Details can be displayed:
   a. Touch the magnifying glass icon next to the cause. The detail page is opened. (>>> *Fig. 4-13*)
   b. Open the descriptions of the cause and solution.
   c. If the message has several possible causes: the magnifying glass icons with arrows can be used to jump to the previous or next detail page.

Message page

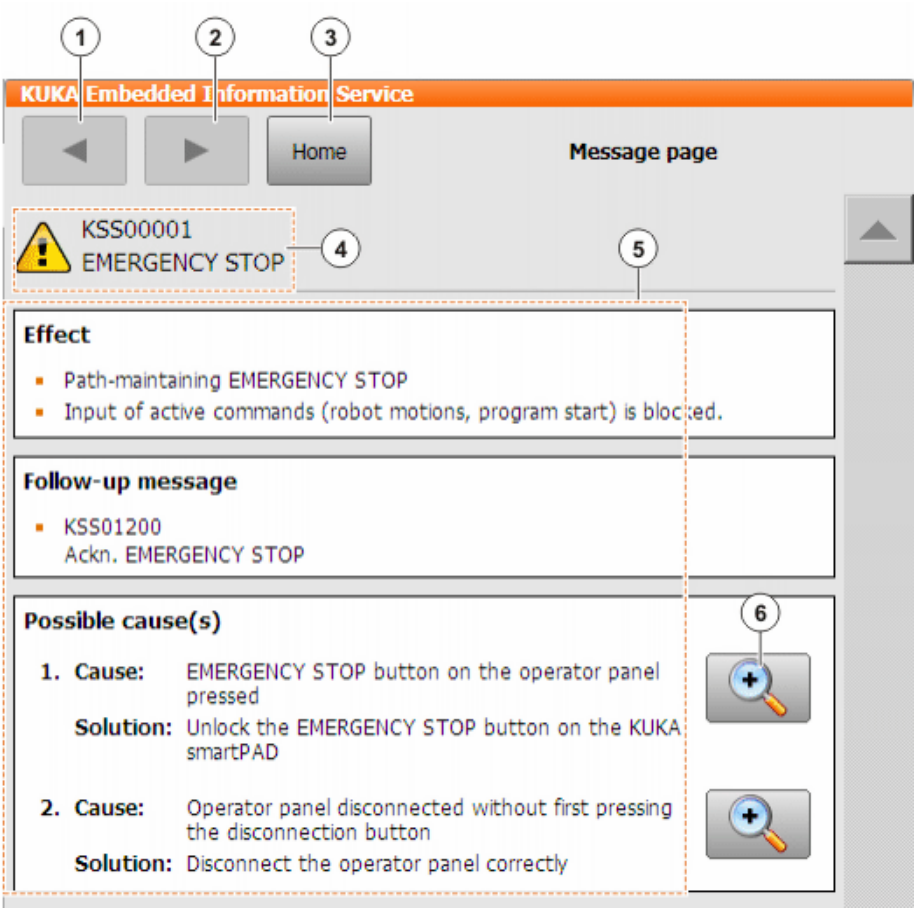


Fig. 4-12: Message page – Example from the KUKA System Software

Item	Description
1	Displays the previous page.
2	This button is only active if the other arrow button has been used to jump to the previous page. This button can then be used to return to the original page.
3	Displays the list with the software modules.
4	Message number and text
5	Information about the message There may be less information available than in the example.
6	Displays details about this cause/solution. (>>> *Fig. 4-13*)

Detail page

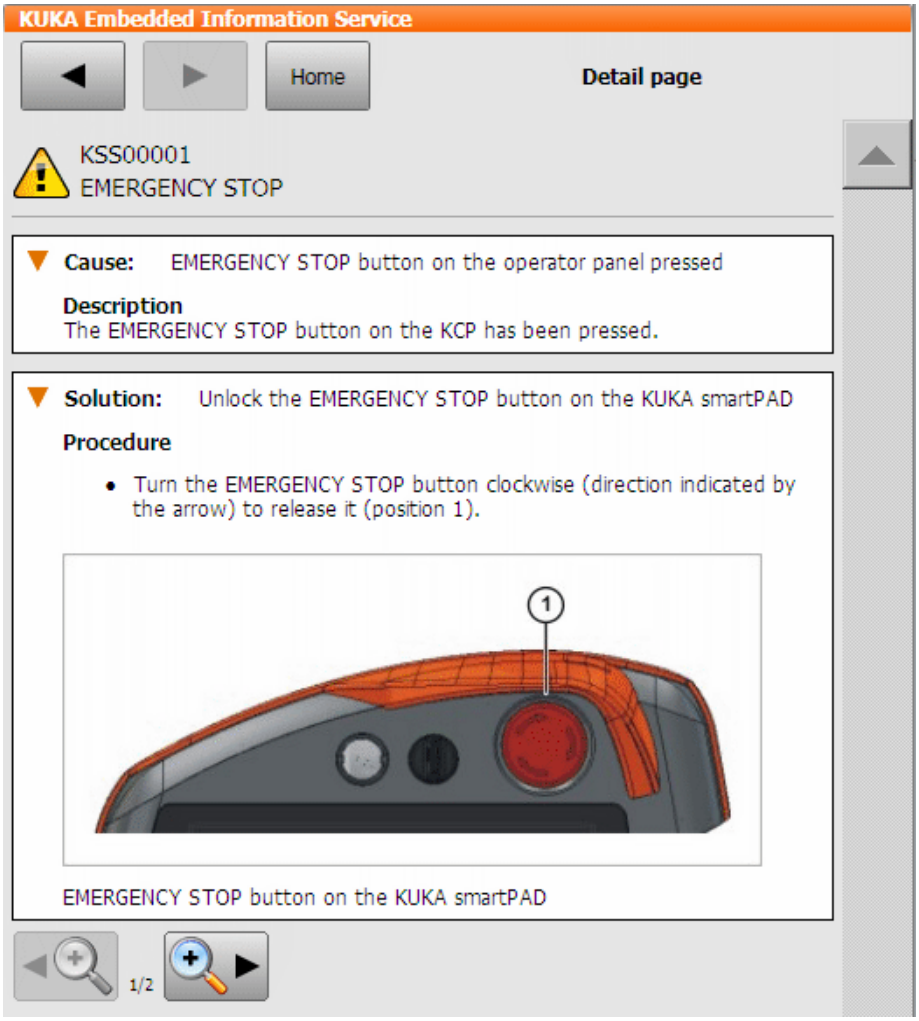


Fig. 4-13: Detail page – Example from the KUKA System Software

4.12 Changing user group

Procedure

1. Open the window for changing the user group:
   - Touch the **User group** icon on the smartHMI.

     Fig. 4-14: User group icon

   - Or: In the main menu, select **Configuration** > **User group**.
2. Change the user group:
   - Press **Default** to switch to the default user group (= **Operator**).
   - To switch to a different user group, select the desired user group. Then enter the password and confirm with **Log on**.

4.13 User groups

Default user group

The default user group is **Operator**. It is the only user group which does not require a password.

- Following a restart, the default user group is automatically selected.
- If the mode is switched to AUT or AUT EXT, the robot controller automatically switches to the default user group for safety reasons. If a different user group is desired, it is only subsequently possible to switch to it.
- If no actions are carried out on the user interface within 5 minutes, the robot controller automatically switches to the default user group for safety reasons.

Current user group

The number of white segments in the **User group** icon on the smartHMI indicates which user group is currently selected.

Example: 3 white segments = **Expert**

User groups

All user groups besides the default user group are password-protected. The default password for all groups is “kuka”. The password can be changed for each group.

(>>> *6.10 "Changing the password" Page 219*)

Each user group has all the rights of the group below plus additional rights.

User group/ Number of segments		Standard rights
Administrator	6	The “Administrator” user group may perform all functions (including those of the safety systems).
Safety maintenance technician	5	The “Safety maintenance” user group may carry out functions required for the start-up of the system (incl. for safety equipment).
Safety recovery technician	4	The “Safety recovery” user group may carry out functions required for the maintenance of systems (incl. for safety equipment). The user rights of the safety recovery technician are restricted by the installation of a safety option. **Note:** Information can be found in the documentation of the safety options, e.g. SafeOperation.
Expert	3	The “Expert” user group may carry out functions which require expert knowledge.
User	2	The “User” user group may carry out functions which are required for the normal operation of the robot.

User group/ Number of segments		Standard rights
Operator No password	1	Very limited rights: The operator may not carry out functions that permanently modify the system.

Administering rights

- The administrator can modify which rights a user group has.
- Every user can display which rights a user group currently has.

(>>> *6.11 "Displaying and modifying user rights" Page 219*)

4.14 Changing operating mode

Do not change the operating mode while a program is running. If the operating mode is changed during program execution, the industrial robot is stopped with a safety stop 2.

Precondition

- The robot controller is not executing a program.
- If the mode selector switch is the variant with a key: the key is inserted in the switch.

Procedure

1. Turn the mode selector switch on the smartPAD. The connection manager is displayed.
2. Select the operating mode. (>>> *3.5.3 "Selecting the operating mode" Page 30*)
3. Return the mode selector switch to its original position.

   The selected operating mode is displayed in the status bar of the smartPAD.

Operating mode	Use	Velocities
T1	For test operation, programming and teaching	• Program verification: Programmed velocity, maximum 250 mm/s • Jog mode: Jog velocity, maximum 250 mm/s
T2	For test operation	• Program verification: Programmed velocity • Jog mode: Not possible
AUT	For industrial robots without higher-level controllers	• Program operation: Programmed velocity • Jog mode: Not possible
AUT EXT	For industrial robots with higher-level controllers, e.g. PLC	• Program operation: Programmed velocity • Jog mode: Not possible

4.15 Coordinate systems

Overview

The following Cartesian coordinate systems are defined in the robot controller:

- WORLD
- ROBROOT
- BASE
- TOOL

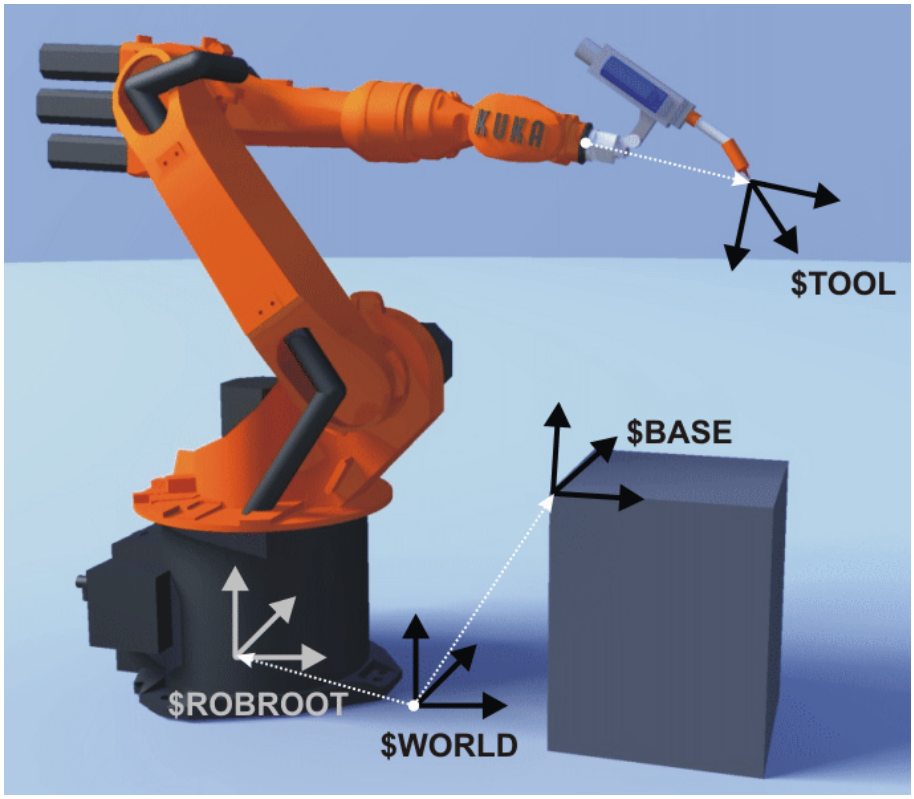


Fig. 4-15: Overview of coordinate systems

Description

WORLD

The WORLD coordinate system is a permanently defined Cartesian coordinate system. It is the root coordinate system for the ROBROOT and BASE coordinate systems.

By default, the WORLD coordinate system is located at the robot base.

ROBROOT

The ROBROOT coordinate system is a Cartesian coordinate system, which is always located at the robot base. It defines the position of the robot relative to the WORLD coordinate system.

By default, the ROBROOT coordinate system is identical to the WORLD coordinate system. $ROBROOT allows the definition of an offset of the robot relative to the WORLD coordinate system.

BASE

The BASE coordinate system is a Cartesian coordinate system that defines the position of the workpiece. It is relative to the WORLD coordinate system.

By default, the BASE coordinate system is identical to the WORLD coordinate system. It is offset to the workpiece by the user.

(>>> *5.15.4 "Introduction to BASE calibration" Page 172*)

TOOL

The TOOL coordinate system is a Cartesian coordinate system which is located at the tool center point.

As standard, the origin of the TOOL coordinate system is located at the flange center point. (In this case it is called the FLANGE coordinate system.) The TOOL coordinate system is offset to the tool center point by the user.

(>>> *5.15.2 "Introduction to TOOL calibration" Page 170*)

Angles of rotation of the robot coordinate systems

Angle	Rotation about axis
Angle A	Rotation about the Z axis
Angle B	Rotation about the Y axis
Angle C	Rotation about the X axis

4.16 Jogging the robot

Description

There are 2 ways of jogging the robot:

- Cartesian jogging

  The TCP is jogged in the positive or negative direction along the axes of a coordinate system.
- Axis-specific jogging

  Each axis can be moved individually in the positive or negative direction.

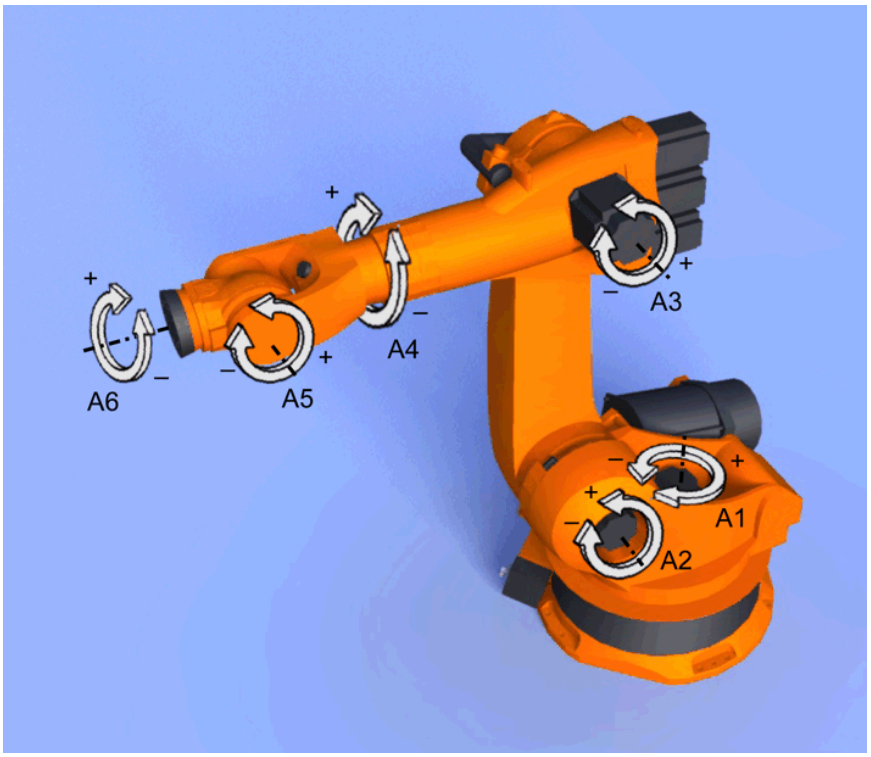


Fig. 4-16: Axis-specific jogging

There are 2 operator control elements that can be used for jogging the robot:

- Jog keys
- Space Mouse

While the robot is being jogged using the keys, the Space Mouse is disabled until the robot comes to a standstill. While the Space Mouse is actuated, the keys are disabled.

Overview

	Cartesian jogging	Axis-specific jogging
Jog keys	(>>> *4.16.5 "Cartesian jogging with the jog keys" Page 88*)	(>>> *4.16.4 "Axis-specific jogging with the jog keys" Page 87*)
Space Mouse	(>>> *4.16.8 "Cartesian jogging with the Space Mouse" Page 91*)	Axis-specific jogging with the Space Mouse is possible, but is not recommended.

4.16.1 "Jog options" window

Description

All parameters for jogging the robot can be set in the **Jogging Options** window.

Procedure

To open the **Jogging options** window:

1. Open a status indicator on the smartHMI, e.g. the **Cur. tool/base** status indicator.

   (Not possible for the **Submit interpreter**, **Drives** and **Robot interpreter** status indicators.)

   A window opens.
2. Press **Options**. The **Jogging Options** window is opened.

For most parameters, it is not necessary to open the **Jogging Options** window. They can be set directly via the smartHMI status indicators.

4.16.1.1 General tab

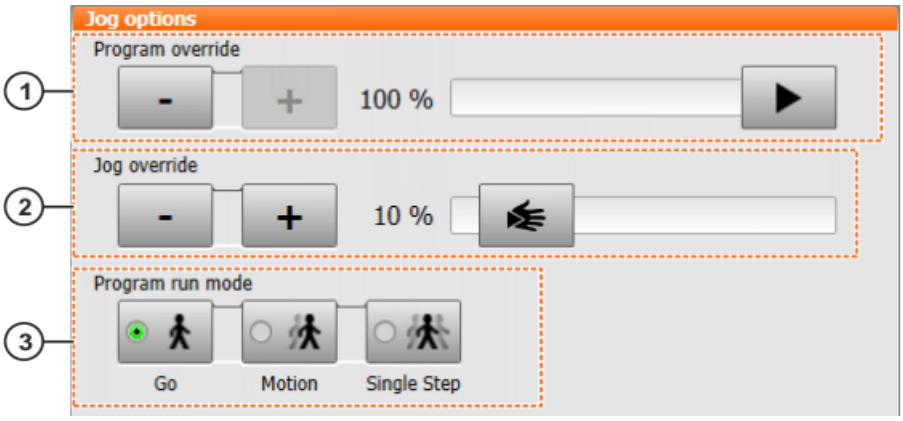


Fig. 4-17: General tab

Description

Item	Description
1	Set the program override. (>>> *8.5 "Setting the program override" Page 331*)
2	Set the jog override. (>>> *4.16.2 "Setting the jog override" Page 86*)
3	Select the program run mode. (>>> *8.2 "Program run modes" Page 327*)

4.16.1.2 Keys tab

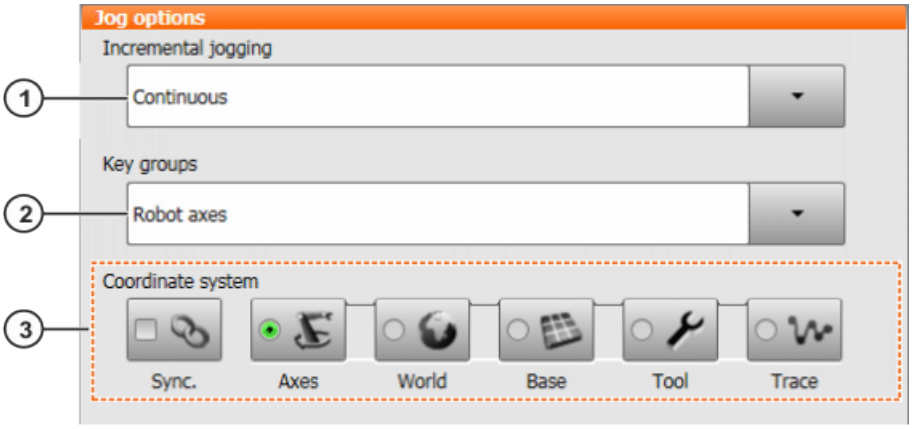


Fig. 4-18: Keys tab

Required user rights for modifications: Function group **General jog settings**

Description

Item	Description
1	Incremental jogging (>>> *4.16.9 "Incremental jogging" Page 91*) The setting automatically changes to **Continuous** (if not already set) if the setting **Track** under **Coordinate system** is selected. If **Track** is deselected again, the setting changes back to the original value.
2	Select kinematics group. The kinematics group defines the axes to which the jog keys refer. The default setting is **Robot axes** (= A1 … A6). Depending on the system configuration, other kinematics groups may be available for selection. (>>> *4.17 "Jogging external axes" Page 95*) The setting automatically changes to **No selection** if the setting **Track** under **Coordinate system** is selected. If **Track** is deselected again, the setting changes back to the original value.

Item	Description
3	Select the reference system for jogging with the jog keys: • **Axes**, **World**, **Base** or **Tool** • **Track**: Most recently executed motions (>>> *4.16.10 "Backward motion using the jog keys" Page 92*) Check box **Sync.**: • Check box not active (default): On the **Keys** and **Mouse** tabs, different coordinate systems can be selected. • Check box active: If the coordinate system on the **Keys** tab is changed, the setting on the **Mouse** tab is adapted automatically and vice versa. Exception: If the **Track** setting is selected on the **Keys** tab, nothing changes on the **Mouse** tab.

4.16.1.3 Mouse tab

Fig. 4-19: Mouse tab

Description

Item	Description
1	Configure the Space Mouse (>>> *4.16.6 "Configuring the Space Mouse" Page 88*) Required user rights: Function group **General configuration**
2	Select the coordinate system for jogging with the Space Mouse: • **Axes**, **World**, **Base** or **Tool** Check box **Sync.**: • Check box not active (default): On the **Keys** and **Mouse** tabs, different coordinate systems can be selected. • Check box active: On the **Keys** and **Mouse** tabs, only one coordinate system can be selected, which is the same in each case. If the coordinate system is changed on one tab, the setting on the other is adapted automatically. Required user rights: Function group **General jog settings**

4.16.1.4 KCP pos. tab

Fig. 4-20: KCP pos. tab

Description

Item	Description
1	(>>> *4.16.7 "Defining the alignment of the Space Mouse" Page 90*)

4.16.1.5 Cur. tool/base tab

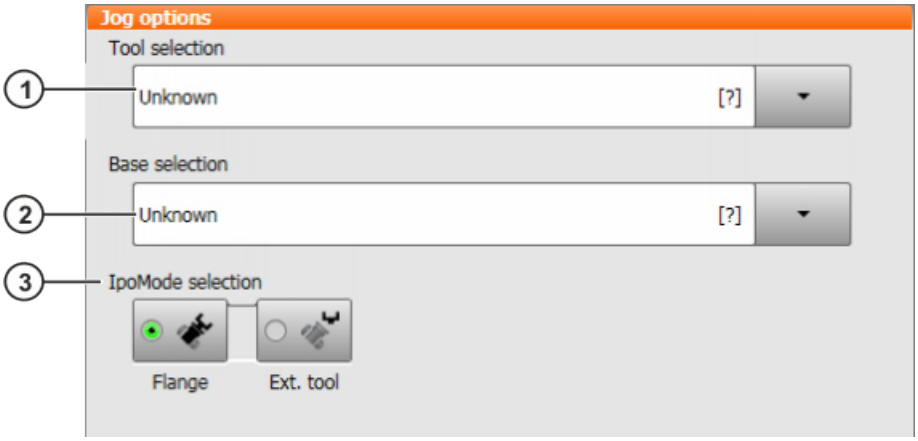


Fig. 4-21: Cur. tool/base tab

Required user rights for modifications: Function group **General jog settings**

Description

Item	Description
1	The current tool is displayed here. A different tool can be selected. (>>> *4.16.3 "Selecting the tool and base" Page 87*) The display **Unknown [?]** means that no tool has yet been calibrated.
2	The current base is displayed here. A different base can be selected. (>>> *4.16.3 "Selecting the tool and base" Page 87*) The display **Unknown [?]** means that no base has yet been calibrated.
3	Select the interpolation mode: • **Flange**: The tool is mounted on the mounting flange. • **Ext. tool**: The tool is a fixed tool.

4.16.1.6 Collision detection tab

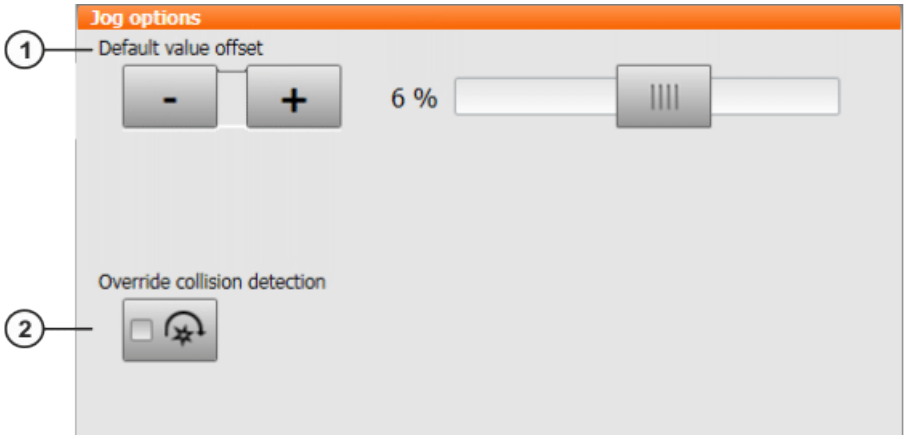


Fig. 4-22: Collision detection tab

Description

Item	Description
1	The sensitivity of the collision detection can be changed here. Changes can be made using either the plus/minus keys or the slider. The percentage value refers to the values under **Configuration** > **Collision detection** > **Jogging configuration**, in the **Default values** column. • Value 0%: no change compared with **Default values** • Negative value: Greater sensitivity, i.e. the detection is triggered more easily • Positive value: Less sensitivity Required user rights: Function group **General jog settings**
2	Following a collision, the forces and torques acting on the robot axes may be so great that the detection function permanently prevents resumption of motion. The user must safely retract the robot by hand, i.e. move it out of the collision position. To enable this, the user can override the collision detection. The override remains in place until canceled by the user. • Check box active: Collision detection is overridden. The robot can be moved out of the collision position. The following message is displayed: *Collision detection deactivated for jogging* • Check box not active: Collision detection is not overridden. Required user rights: Function group **Critical jog settings** **Note:** For safe retraction of the robot, there is also the jog mode **Track**. **Track** is to be preferred. Only use **Override collision detection** if **Track** cannot be used, e.g. if the robot is jammed following the collision.

4.16.2 Setting the jog override

Description

Jog override determines the velocity of the robot during jogging. The velocity actually achieved by the robot with a jog override setting of 100% depends on various factors, including the robot type. The velocity cannot exceed 250 mm/s however.

Precondition

- User rights: Function group **General jog settings**

Procedure

1. Touch the status indicator **Overrides**. The **Overrides** window opens.

Fig. 4-23: Overrides status indicator

2. Set the desired jog override. It can be set using either the plus/minus keys or by means of the slider.

- Plus/minus keys: The value can be set to 100%, 75%, 50%, 30%, 10%, 5%, 3%, 1%.
- Slider: The override can be adjusted in 1 % steps.

3. Touch the status indicator **Overrides** again. (Or touch the area outside the window.)

   The window closes and the selected override value is applied.

Alternative procedure

Alternatively, the override can be set using the plus/minus key on the lower right-hand side of the smartPAD.

The value can be set to 100%, 75%, 50%, 30%, 10%, 5%, 3%, 1%.

4.16.3 Selecting the tool and base

Description

One tool (TOOL coordinate system) and one base (BASE coordinate system) must be selected for Cartesian jogging.

Precondition

- User rights: Function group **General jog settings**

Procedure

1. Touch the status indicator **Cur. tool/base**.

Fig. 4-24: Cur. tool/base status indicator

   The **Cur. tool/base** window opens.

2. Select the desired tool and base.

   The window closes automatically and the selection is applied.

4.16.4 Axis-specific jogging with the jog keys

Precondition

- User rights: Function group **Jogging with the jog keys**
- **Axes** is selected as the coordinate system for jogging with the keys.
- The desired jog override is set.
- T1 mode

Procedure

1. Press and hold down the enabling switch.

   Axes A1 to A6 are displayed next to the jog keys.

2. Press the plus or minus jog key to move an axis in the positive or negative direction.

> The position of the robot during jogging can be displayed: select **Display** > **Actual position** in the main menu.

4.16.5 Cartesian jogging with the jog keys

Precondition

- User rights: Function group **Jogging with the jog keys**
- **World**, **Base** or **Tool** is selected as the coordinate system for jogging with the keys.
- The desired jog override is set.
- Tool and base have been selected.
- T1 mode

Procedure

1. Press and hold down the enabling switch.

   The following designations are displayed next to the jog keys:

   - **X**, **Y**, **Z**: for the linear motions along the axes of the selected coordinate system
   - **A**, **B**, **C**: for the rotational motions about the axes of the selected coordinate system
2. Press the plus or minus jog key to move the robot in the positive or negative direction.

> The position of the robot during jogging can be displayed: select **Display** > **Actual position** in the main menu.

4.16.6 Configuring the Space Mouse

Precondition

- User rights: Function group **General configuration**

Procedure

1. Open the **Jog options** window and select the **Mouse** tab.
   (>>> *4.16.1 ""Jog options" window" Page 81*)
2. Group **Mouse settings**:
   - **Dominant** check box:
     Activate or deactivate dominant mode as desired.
   - **6D**/**XYZ**/**ABC** option box:
     Select whether the TCP is to be moved using translational motions, rotational motions, or both.
3. Close the **Jog options** window.

Description

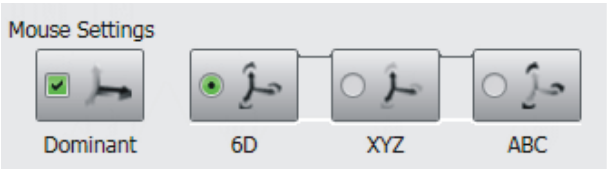


Fig. 4-25: Mouse settings

Dominant check box:

Depending on the dominant mode, the Space Mouse can be used to move just one axis or several axes simultaneously.

Check box	Description
Active	Dominant mode is activated. Only the coordinate axis with the greatest deflection of the Space Mouse is moved.
Inactive	Dominant mode is deactivated. Depending on the axis selection, either 3 or 6 axes can be moved simultaneously.

Option	Description
6D	The robot can be moved by pulling, pushing, rotating or tilting the Space Mouse. The following motions are possible with Cartesian jogging: • Translational motions in the X, Y and Z directions • Rotational motions about the X, Y and Z axes
XYZ	The robot can only be moved by pulling or pushing the Space Mouse. The following motions are possible with Cartesian jogging: • Translational motions in the X, Y and Z directions
ABC	The robot can only be moved by rotating or tilting the Space Mouse. The following motions are possible with Cartesian jogging: • Rotational motions about the X, Y and Z axes

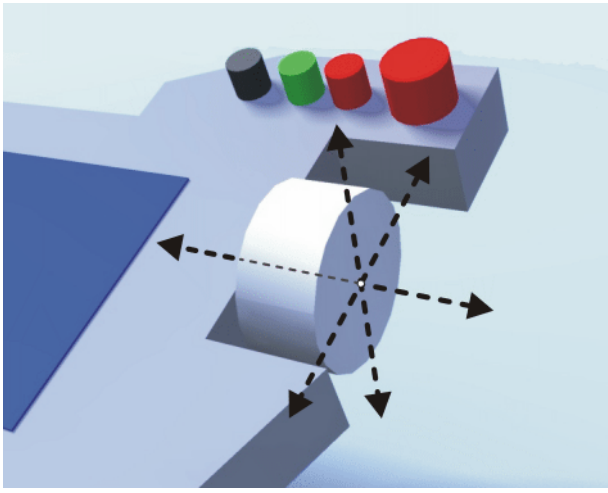


Fig. 4-26: Pushing and pulling the Space Mouse

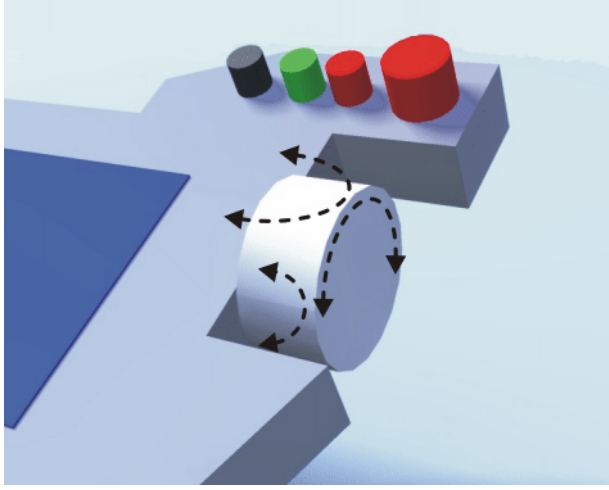


Fig. 4-27: Rotating and tilting the Space Mouse

4.16.7 Defining the alignment of the Space Mouse

Description

The functioning of the Space Mouse can be adapted to the location of the user so that the motion direction of the TCP corresponds to the deflection of the Space Mouse.

The location of the user is specified in degrees. The reference point for the specification in degrees is the junction box on the base frame. The position of the robot arm or axes is irrelevant.

Default setting: 0°. This corresponds to a user standing opposite the junction box.

Switching to Automatic External mode automatically resets the alignment of the Space Mouse to 0°.

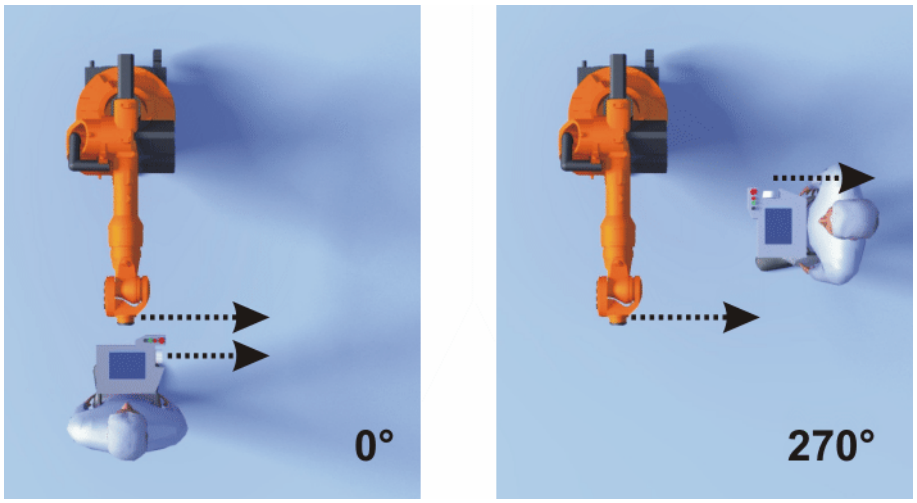


Fig. 4-28: 6D mouse: 0° and 270°

Precondition

- User rights: Function group **General jog settings**
- T1 mode

Procedure

1. Open the **Jog options** window and select the **KCP pos.** tab.

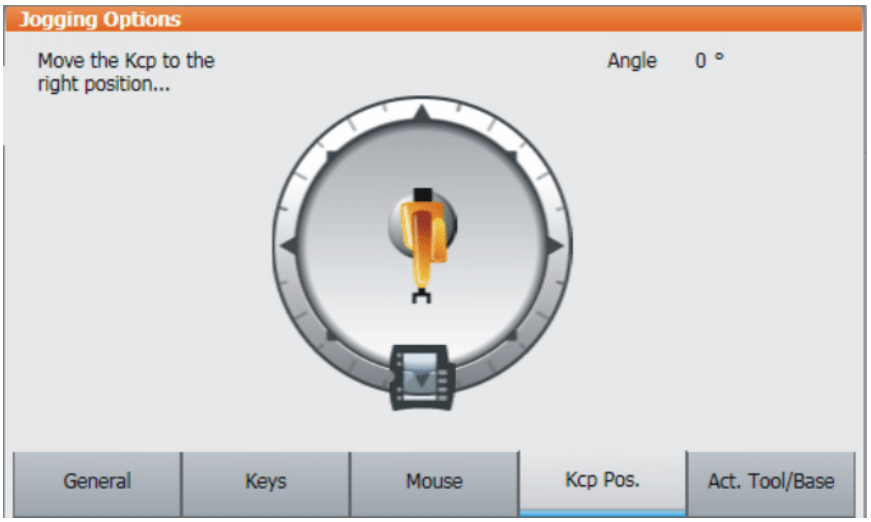


Fig. 4-29: Defining the alignment of the Space Mouse

2. Drag the smartPAD to the position corresponding to the location of the user (in 45° steps).
3. Close the **Jog options** window.

4.16.8 Cartesian jogging with the Space Mouse

Precondition

- User rights: Function group **Jogging using the 6D mouse**
- **World**, **Base** or **Tool** is selected as the coordinate system for the Space Mouse.
- The desired jog override is set.
- Tool and base have been selected.
- T1 mode
- The Space Mouse is configured.
- The alignment of the Space Mouse has been defined.

Procedure

1. Press and hold down the enabling switch.
2. Move the robot in the desired direction using the Space Mouse.

The position of the robot during jogging can be displayed: select **Display** > **Actual position** in the main menu.

4.16.9 Incremental jogging

Description

Incremental jogging makes it possible to move the robot a defined distance, e.g. 10 mm or 3°. The robot then stops by itself.

- Incremental jogging can be activated for jogging with the jog keys. Exception: It cannot be activated for backward motion.
- Incremental jogging is not possible in the case of jogging with the Space Mouse.

Areas of application:

- Positioning of equidistant points
- Moving a defined distance away from a position, e.g. in the event of a fault
- Mastering with the dial gauge

The following options are available:

Setting	Description
Continuous	Incremental jogging is deactivated.
100 mm / 10°	1 increment = 100 mm or 10°
10mm / 3°	1 increment = 10 mm or 3°
1mm / 1°	1 increment = 1 mm or 1°
0.1mm / 0.005°	1 increment = 0.1 mm or 0.005°

Increments in mm:

- Valid for Cartesian jogging in the X, Y or Z direction.

Increments in degrees:

- Valid for Cartesian jogging in the A, B or C direction.
- Valid for axis-specific jogging.

Precondition

- User rights of the following function groups:
  - **General jog settings**
  - **Jogging with the jog keys**
- The jog option **Track** is not active.
- T1 mode

Procedure

1. Select the size of the increment in the status bar.
2. Jog the robot using the jog keys. Jogging can be Cartesian or axis-specific.

   Once the set increment has been reached, the robot stops.

If the robot motion is interrupted, e.g. by releasing the enabling switch, the interrupted increment is not resumed with the next motion; a new increment is started instead.

4.16.10 Backward motion using the jog keys

In addition to backward motion using the jog keys, there is another option for backward motion. Information about it can be found here: (>>> *8.13 "Backward motion using the Start backwards key" Page 336*)
An overview of the most important differences can be found here: (>>> *8.13.4 "Comparison of “Start backwards”/backwards using the jog keys" Page 341*)

4.16.10.1 Backward motion using the jog keys – Overview

Description

The robot controller records the motions of the robot. The recording serves as a “memory” for the robot and enables it to execute motions backwards.

Backward motion is carried out using the jog keys. The robot is able, however, to execute backwards not only the motions that were originally executed using the jog keys, but virtually all motions. It makes no difference how the original motion sequence was put together, e.g. alternating manual and program motions: in backward motion, there is a single continuous path. It can be stopped at any point, but the original motions cannot be executed backwards individually (motion by motion).

If the original motions had exact positioning points, the robot also stops there during backward motion. Exception: If the path passes through the exact positioning point in a virtually straight line, it does not stop.

The user determines the velocity of the backward motion by means of the jog override. The velocity of the original motion is irrelevant.

The robot is able to execute the recorded path not only backwards, but also backwards and then forwards again. It can, so to speak, move backwards and forwards in the past. It is possible to change direction any number of times.

Motion types

The following motions are recorded and can be executed backwards:

- Manual motions
- Program motions
- BCO runs
- Motions of synchronous and asynchronous axes
- Weave motions
- Interrupt motions
- Motions executed using the Start backwards key

The following motions are not recorded and thus cannot be executed backwards:

- Motions of axes switched to “soft” mode
- The backward motion itself, i.e. the motions executed on the recorded path using the jog keys

BCO

If program motions are executed backwards, this results in loss of BCO. For this reason, when the program is resumed after backward motion, the robot controller performs a BCO run.

4.16.10.2 Recording in buffer

Description

The robot controller records the motions of the robot in a buffer. How far it is possible to move backwards (i.e. how far back the robot’s memory extends) depends on how many motions have been recorded. This, in turn, depends on numerous factors, including the curvature of the path and the robot type. The extent to which backward motion is possible thus depends on the individual case and can vary greatly.

The buffer for backward motion using the jog keys is not the same as the buffer for backward motion using the “Start backwards” key.

The contents of the buffer are deleted or partially deleted in certain situations.

General deletion

The following actions delete the buffer:

- Axis mastering
- Coupling or decoupling external axes

If using KUKA.RoboTeam, the buffer is also deleted in the following cases:

- Jogging with LoadSharing
- Jogging using the LK function
- Jogging with the synchronization type #MotionSync

Deletion after negative position comparison

The buffer contents are retained beyond the following actions:

- **I/O drivers** > **Reconfigure**
- Reboot with **Hibernate** or **Cold start**

Following these actions, the robot controller compares the current robot position with the most recent position in the buffer.

Or, if backward motion has already been carried out before the reboot/ reconfiguration: The robot controller compares the current position with the most recently addressed position on the recorded path.

- If the points are identical, the buffer is retained.
- If the points are not identical, the buffer is deleted.

Deletion on Start from buffer

If the user has moved the robot backwards and then starts another motion, part of the buffer is deleted.

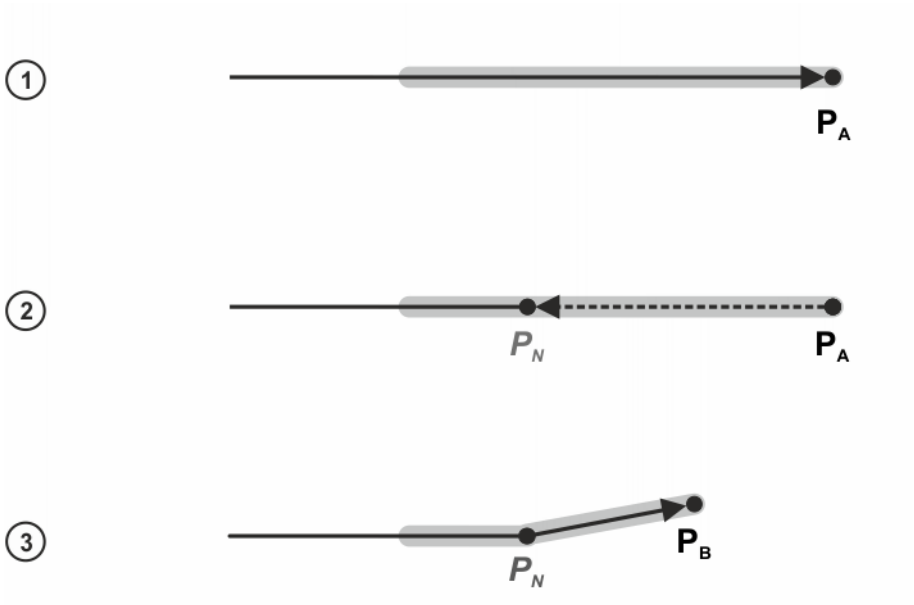


Fig. 4-30: Deletion on Start from buffer

Step	Description
1	Initial situation: The user has moved the robot forwards. The robot is located at a point P_A. The last part of the path to this point is saved in the buffer (thick gray bar).
2	The user now moves the robot backwards (dashed black arrow). The robot is now located at a point P_N in the buffer. At this point in time, the same path is still saved in the buffer as for step 1.
3	At point P_N, the user starts a different motion. This can be any motion other than continued motion along the recorded path (no matter whether backwards or forwards). The robot is now located at a point P_B. • The section from P_N to P_A has been deleted from the buffer. • The new motion from P_N to P_B has been recorded.

4.16.10.3 Executing motions backwards (using jog keys)

Precondition

- User rights of the following function groups:
  - **General jog settings**
  - **Jogging with the jog keys**
- Neither the “Start forwards” nor the “Start backwards” key has been pressed.
- All axes are stationary, including asynchronous axes (if present).
- T1 mode

Procedure

1. Select the option **Track** on the **Keys** tab in the **Jog options** window.
2. Set the jog override.
3. Press and hold down the enabling switch.

   The following icon is displayed next to the uppermost jog key:
4. Press minus on the jog key to move the robot backwards along the recorded path.
5. Press plus to move the robot forwards along the recorded path.

   The robot can be moved alternately forwards and backwards. If the end of the recorded path is reached in either direction, the robot controller displays the following message: *End of recorded path reached.*

> The position of the robot during jogging can be displayed: select **Display** > **Actual position** in the main menu.

4.17 Jogging external axes

Description

External axes must be jogged using the jog keys. They cannot be moved using the Space Mouse.

Precondition

- User rights of the following function groups:
  - **General jog settings**
  - **Jogging with the jog keys**
- T1 mode

Procedure

1. Select the desired kinematics group, e.g. **External axes**, on the **Keys** tab in the **Jog options** window.

   The type and number of kinematics groups available depend on the system configuration.
2. Set jog override.
3. Hold down the enabling switch.

   The axes of the selected kinematics group are displayed next to the jog keys.
4. Press the Plus or Minus jog key to move an axis in the positive or negative direction.

Kinematic groups

Depending on the system configuration, the following kinematics groups may be available.

Kinematics group	Description
Robot axes	The robot axes can be moved using the jog keys. The external axes cannot be jogged.
External axes	All configured external axes (e.g. external axes E1 to E5) can be moved using the jog keys.
NAME / **External Kinematics Group** *n*	The axes of an external kinematics group can be moved using the jog keys. The name is taken from the system variable $ET*n_NAME* (*n* = number of the external kinematic system). If $ET*n_NAME* is empty, the default name **External Kinematics Group** *n* is displayed.
[User-defined kinematics group]	The axes of a user-defined kinematics group can be moved using the jog keys. The name corresponds to the name of the user-defined kinematics group.

4.18 Bypassing workspace monitoring

Description

Workspaces can be configured for a robot. These serve to protect the system. A workspace is always of one of the two following types:

- Space that the robot must not leave

  A monitoring function is triggered if the robot leaves the space.
- Space that the robot must not enter

  A monitoring function is triggered if the robot enters the space.

Exactly what reactions occur when the monitoring function is triggered depends on the configuration.

One possible reaction, for example, is that the robot stops. In this case, the workspace monitoring function must be bypassed or deactivated in order to be able to move the robot back out of the violated space.

Precondition

- User rights: Function group **General configuration**
- T1 or T2 mode

Procedure

1. In the main menu, select **Configuration** > **Miscellaneous** > **Workspace monitoring** > **Override**.
2. Move the robot manually out of the violated space.

   Once the robot has left the violated space, the workspace monitoring is automatically active again.

4.19 Display functions

4.19.1 Displaying the actual position

Procedure

1. In the main menu, select **Display** > **Actual position**. The **Actual position** window opens.

   The most recently selected view is displayed, i.e. either **Cartesian** or **Axis-specific**.
2. To switch to the other view, touch the corresponding button.

The actual position can also be displayed while the robot is moving.

4.19.1.1 Window Actual position, view Cartesian

Information about the current Cartesian position is displayed.

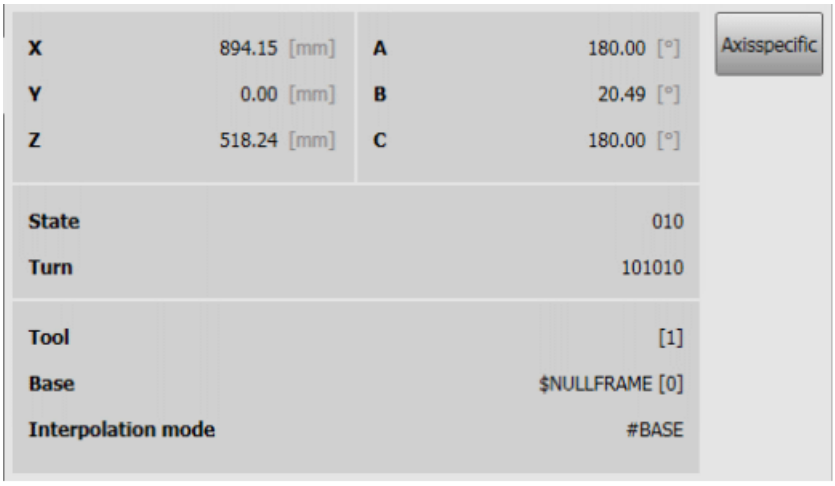


Fig. 4-31: Window Actual position, view Cartesian

Button	Description
Axis-specific	Switches to the view **Axis-specific**

Display boxes	Description
X, **Y**, **Z**	Current position
A, **B**, **C**	Current orientation
Status	Status in binary representation, 3-digit
Turn	Turn in binary representation, 6-digit
Tool **Base**	• Name and number of the valid tool or the valid base or $NULLFRAME[0] • If no tool/no base is valid, e.g. after booting the controller, “**?[-1]**” is displayed. (>>> *"Restriction for Tool/Base" Page 98*)
Interpolation mode	• #BASE: The tool is mounted on the mounting flange. • #TCP: The tool is a fixed tool.

Restriction for Tool/Base

Under certain circumstances, the view **Cartesian** cannot display the valid tool or the valid base, but rather retains the previous value.

This is the case if $TOOL and $BASE have been assigned values directly (X, Y, Z, A, B, C). The view, however, can only display tools and bases defined via a number (TOOL_DATA[…] / BASE_DATA[]).

This case can only occur if a programmer has set the tool or the base in KRL as stated above. Tools and bases selected in inline forms or by means of other elements on the smartHMI are always displayed correctly.

4.19.1.2 Window Actual position, view Axis-specific

The current position of axes A1 to A6 is displayed. If external axes are being used, the position of the external axes is also displayed.

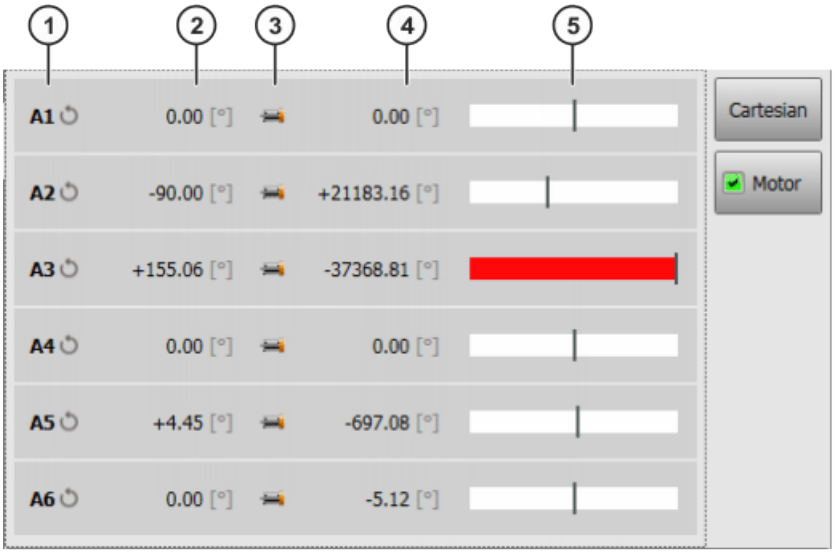


Fig. 4-32: Window Actual position, view Axis-specific

Item	Description
1	Axis name and symbolic representation of the axis type • : rotational • : infinitely rotating • : linear
2	Current axis position
3	“Motor” icon Is only displayed if the check box **Motor** is activated.
4	Current motor angle Is only displayed if the check box **Motor** is activated.
5	In the case of rotational and linear axes: graphical representation of the current position, relative to the permissible axis range • The bar represents the permissible range, i.e. the range between the software limit switches. The left-hand side corresponds to the negative limit switch, the right-hand side to the positive one. The center of the bar corresponds to the center of the permissible range. Example: The negative limit is positioned at 60°, the positive one at 100°. The center of the bar thus corresponds to 20°. • The vertical line represents the position of the axis in the range. • If an axis has reached a limit switch, the bar turns red. The display differs in the case of infinitely rotating axes. (>>> *"Infinitely rotating axes" Page 99*)

Button	Description
Cartesian	Switches to the view **Cartesian**

Check box	Description
Motor	• Check box active: The motor angles of the axes are displayed. • Check box not active: No motor angles are displayed.
Slaves	The check box is only displayed if at least one axis has a slave motor. • Check box active: The name, mode and motor angle of the slave are displayed. • Check box not active: No information about slaves is displayed.

Infinitely rotating axes

Example without display of the motor data:

Fig. 4-33: Infinitely rotating axis, display without motor data

In the example	Description
10	Number of revolutions for this axis
	Graphical representation of the actual axis angle
+77.34	Actual axis angle (modulo 360°) Range: -180° … +180°

Example with display of the motor data:

Fig. 4-34: Infinitely rotating axis, display with motor data

Unlike in the display without motor data, the arrows and the number of revolutions are not displayed here.

4.19.2 Displaying digital inputs/outputs

Procedure

1. In the main menu, select **Display** > **Inputs/outputs**. Then select **Digital inputs** or **Digital outputs**.
2. To display a specific input/output:
   - Press the **Go to** button. The **Go to:** box is displayed.
   - Enter the number and confirm with the Enter key.

   The display jumps to the input/output with this number.

Description

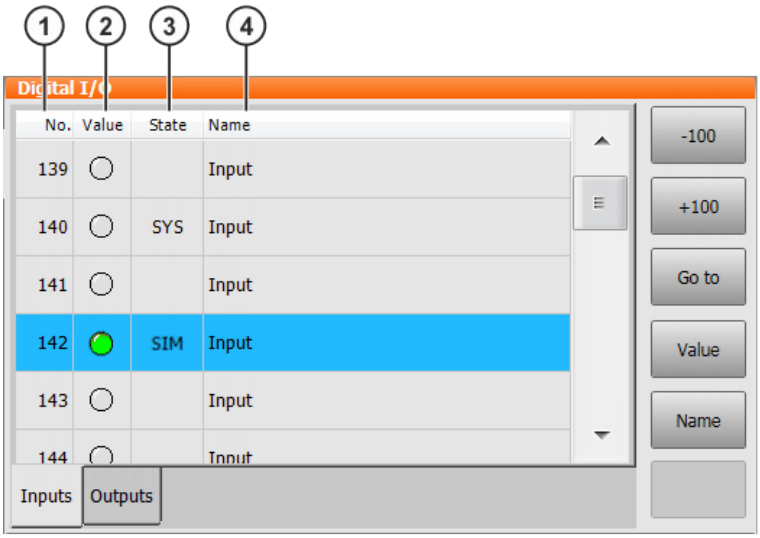


Fig. 4-35: Digital inputs

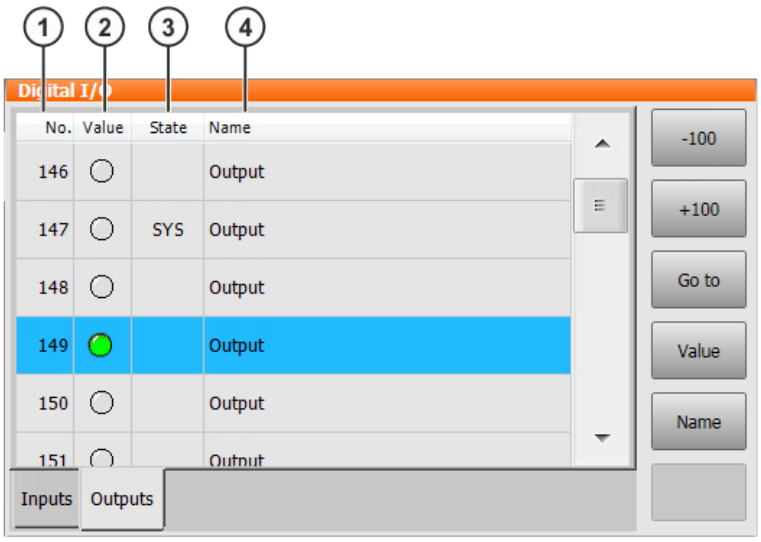


Fig. 4-36: Digital outputs

Item	Description
1	Number of the input/output
2	Value of the input/output. The icon is green if the input or output is TRUE.
3	SIM entry: The input/output is simulated. SYS entry: The value of the input/output is saved in a system variable. This input/output is write-protected.
4	Name of the input/output

Button	Description
-100	Toggles back 100 inputs or outputs in the display.
+100	Toggles forward 100 inputs or outputs in the display.
Go to	The number of the input or output being searched for can be entered.
Value	Toggles the selected input/output between TRUE and FALSE. Precondition: The enabling switch is pressed. • **Value** is not available in AUT EXT mode. • **Value** is only available for inputs if simulation is activated.
Name	The name of the selected input or output can be changed.
Required user rights for modifications via **Value**, **Name**: Function group **General configuration**	

4.19.3 Displaying analog inputs/outputs

Procedure

1. In the main menu, select **Display** > **Inputs/outputs**. Then select **Analog inputs** or **Analog outputs**.
2. To display a specific input/output:

- Press the **Go to** button. The **Go to:** box is displayed.
- Enter the number and confirm with the Enter key.

The display jumps to the input/output with this number.

Button	Description
Go to	The number of the input or output being searched for can be entered.
Value	A voltage can be entered for the selected output. • **-10 … 10 V** This button is only available for outputs.
Name	The name of the selected input or output can be changed.
Required user rights for modifications via **Value**, **Name**: Function group **General configuration**	

4.19.4 Displaying inputs/outputs for Automatic External

Procedure

- In the main menu, select **Display** > **Inputs/outputs** > **Automatic External**.

Description

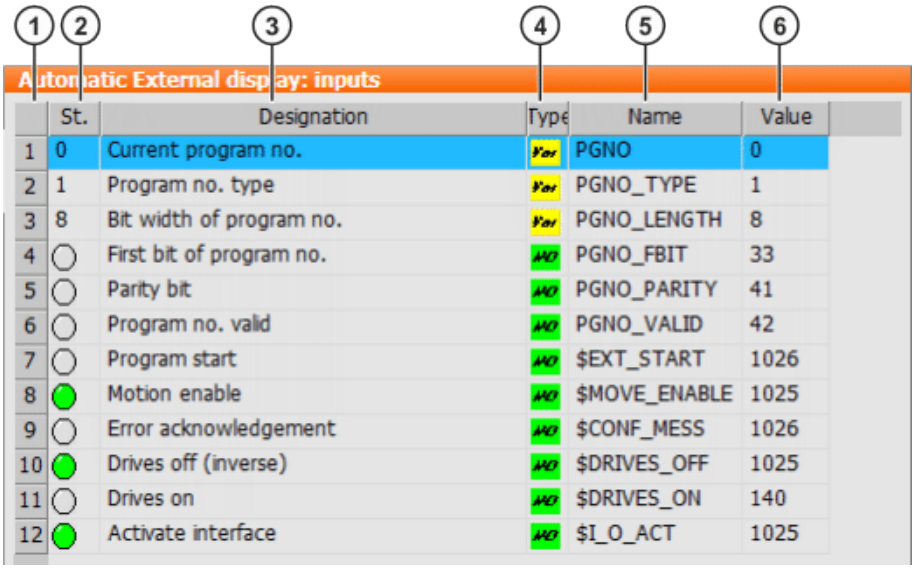


Fig. 4-37: Automatic External inputs (detail view)

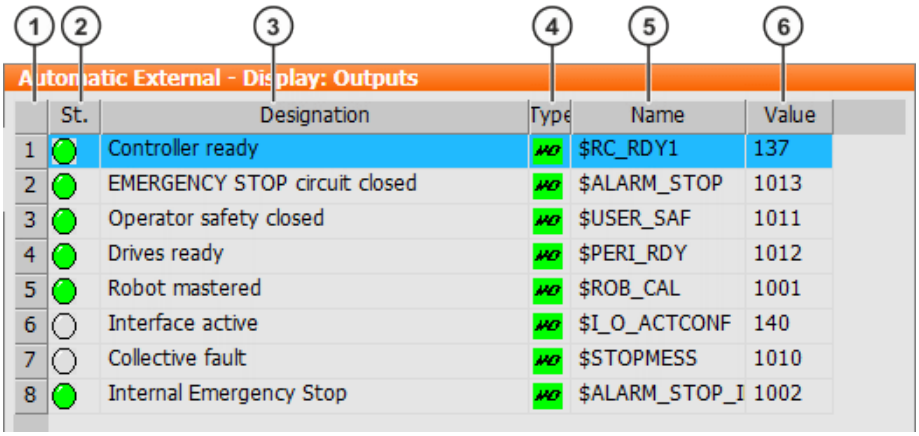


Fig. 4-38: Automatic External outputs (detail view)

Item	Description
1	Number
2	Status • **Gray**: inactive (FALSE) • **Green**: active (TRUE)
3	Long text name of the input/output
4	Type • **Green**: input/output • **Yellow**: variable or system variable ($...)
5	Name of the signal or variable
6	Input/output number or channel number

Columns 4, 5 and 6 are only displayed if **Details** has been pressed.

Button	Description
Configure	Switches to the configuration of the Automatic External interface. (>>> *6.19.2 "Configuring Automatic External inputs/outputs" Page 240*)
Inputs/Outputs	Toggles between the windows for inputs and outputs.
Details/Normal	Toggles between the **Details** and **Normal** views.

4.19.5 Displaying and modifying the value of a variable

This function is also called “variable correction”.

Precondition

To modify a variable:

- User rights: Function group **General configuration**

Procedure

1. In the main menu, select **Display** > **Variable** > **Single**.
   The **Variable display – Single** window opens.

2. Enter the name of the variable in the **Name** box and confirm with the Enter key.
3. If a program has been selected, it is automatically entered in the **Module** box.

   If a variable from a different program is to be displayed, enter the program as follows:

   /R1/*Program name*

   - Do not specify a folder between /R1/ and the program name. Do not add a file extension to the file name.
   - In the case of system variables, no program needs to be specified in the **Module** box.
4. The current value of the variable is displayed in the **Current value** box. If nothing is displayed, no value has yet been assigned to the variable.

   To modify the variable:
5. Enter the desired value in the **New value** box.
6. Press the **Set value** button. The new value is displayed in the **Current value** box.

Description

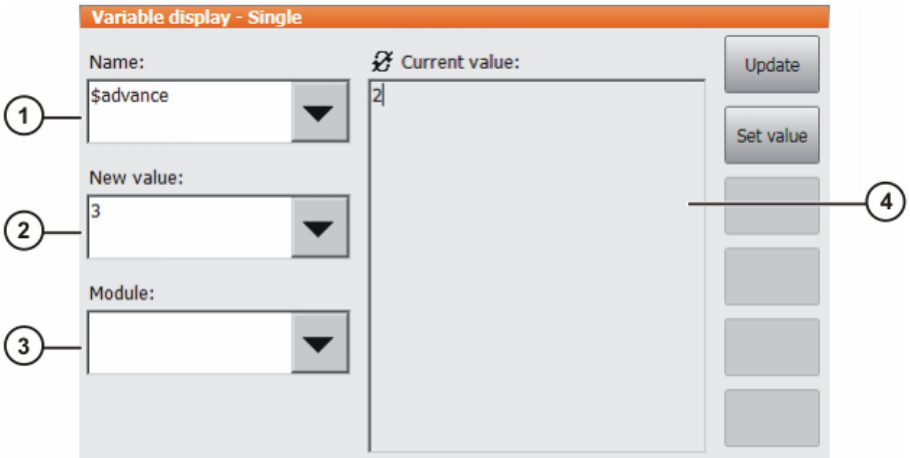


Fig. 4-39: Variable Overview - Single window

Item	Description
1	Name of the variable to be modified.
2	New value to be assigned to the variable.
3	Program in which the search for the variable is to be carried out. In the case of system variables, the **Module** box is irrelevant.
4	This box has two states: • : The displayed value is not refreshed automatically. • : The displayed value is refreshed automatically. Switching between the states: • Press **Refresh**.

4.19.6 Displaying the state of a variable

Description

Variables can have the following states:

- UNKNOWN: The variable is unknown.
- DECLARED: The variable is declared.
- INITIALIZED: The variable is initialized.

Procedure

1. In the main menu, select **Display** > **Variable** > **Single**.

   The **Variable display - Single** window is opened.
2. In the **Name** box, enter: =VARSTATE("*name*")

   name = name of the variable whose state is to be displayed.
3. If a program has been selected, it is automatically entered in the **Module** box.

   If a variable from a different program is to be displayed, enter the program as follows:

   /R1/*Program name*

   - Do not specify a folder between /R1/ and the program name. Do not add a file extension to the file name.
   - In the case of system variables, no program needs to be specified in the **Module** box.
4. Press **Update**.

   The current state of the variable is displayed in the **Current value** box.

4.19.7 Displaying the variable overview and modifying variables

In the variable overview, variables are displayed in groups. The variables can be modified.

The number of groups and which variables they contain are defined in the configuration. By default, the variable overview is empty.

(>>> *6.9 "Configuring the variable overview" Page 217*)

Precondition

The minimum user group required for displaying and/or modifying variables depends on the configuration settings for the variable overview.

Procedure

1. In the main menu, select **Display** > **Variable** > **Overview** > **Display**.

   The **Variable overview – Display** window is opened.
2. Select the desired group.
3. Select the cell to be modified. Carry out modification using the buttons.
4. Press **OK** to save the change and close the window.

Description

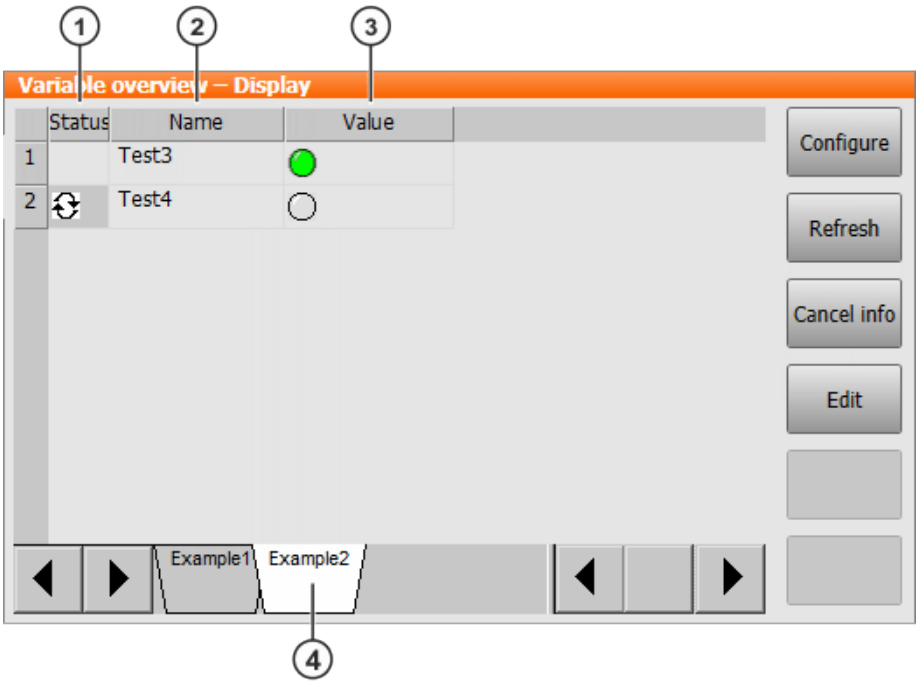


Fig. 4-40: “Variable overview – Display” window

Item	Description
1	Arrow symbol: If the value of the variable changes, the display is automatically refreshed. No arrow symbol: the display is not automatically refreshed.
2	Descriptive name
3	Value of the variable. In the case of inputs/outputs, the state is indicated: • **Gray**: inactive (FALSE) • **Green**: active (TRUE)
4	There is one tab per group.

The following buttons are available:

Button	Description
Configure	Switches to the configuration of the variable overview. (>>> *6.9 "Configuring the variable overview" Page 217*)
Refresh	Refreshes the display.
Cancel info	Deactivates the automatic refreshing function.
Start info	Activates the automatic refreshing function. A maximum of 12 variables per group can be refreshed automatically.

Button	Description
Edit	Different functions, depending on the column in which a cell is selected: • **Status** column: activates or deactivates automatic refreshing. • **Name** column: switches the cell to edit mode so that the name can be modified. • **Value** column: switches the cell to edit mode so that the entry can be modified. In the case of inputs/outputs, **Edit** changes the status (TRUE/FALSE). The **Edit** button is only available in the user group “User” if it has been enabled in the configuration of the variable overview. **Note:** The values of write-protected variables cannot be changed.

4.19.8 Displaying cyclical flags

Procedure

1. In the main menu, select **Display** > **Variable** > **Cyclical flags**. The **Cyclical flags** window is opened.
2. To display a specific flag:
   - Click on the **Go to** button. The **Go to:** box is displayed.
   - Enter the number and confirm with the Enter key.

   The display jumps to the flag with this number.

Description

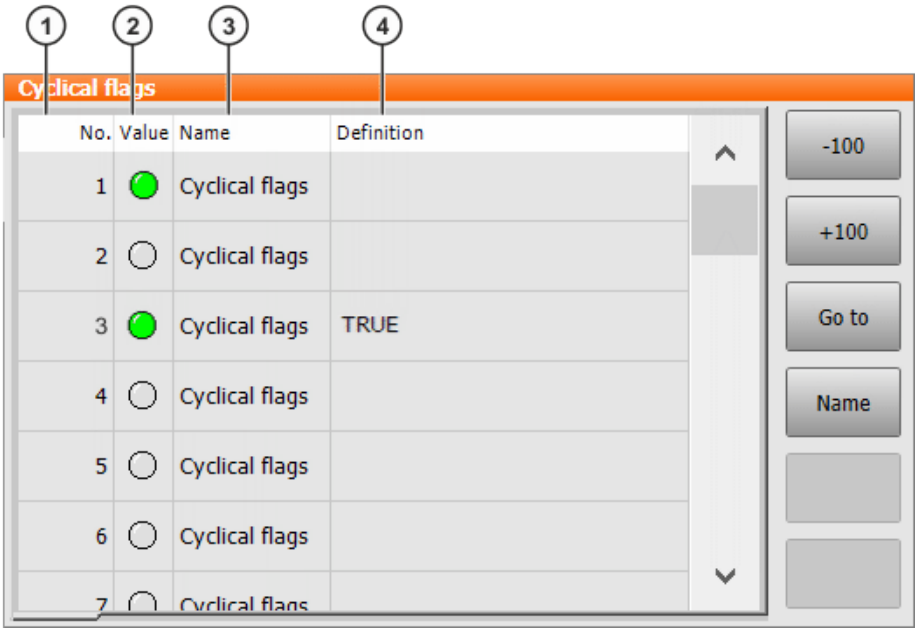


Fig. 4-41: Cyclical flags

Item	Description
1	Flag number
2	Value of the flag. The icon is green if a flag is set.
3	Name of the flag
4	The conditions linked to the setting of a cyclical flag are indicated here.

Button	Description
-100	Toggles back 100 flags in the display.
+100	Toggles forward 100 flags in the display.
Go to	The number of the flag being searched for can be entered.
Name	The name of the selected flag can be modified. Required user rights for modifications: Function group **General configuration**

4.19.9 Displaying flags

Procedure

1. In the main menu, select **Display** > **Variable** > **Flags**. The **Flags** window is opened.
2. To display a specific flag:
   - Click on the **Go to** button. The **Go to:** box is displayed.
   - Enter the number and confirm with the Enter key.

   The display jumps to the flag with this number.

Description

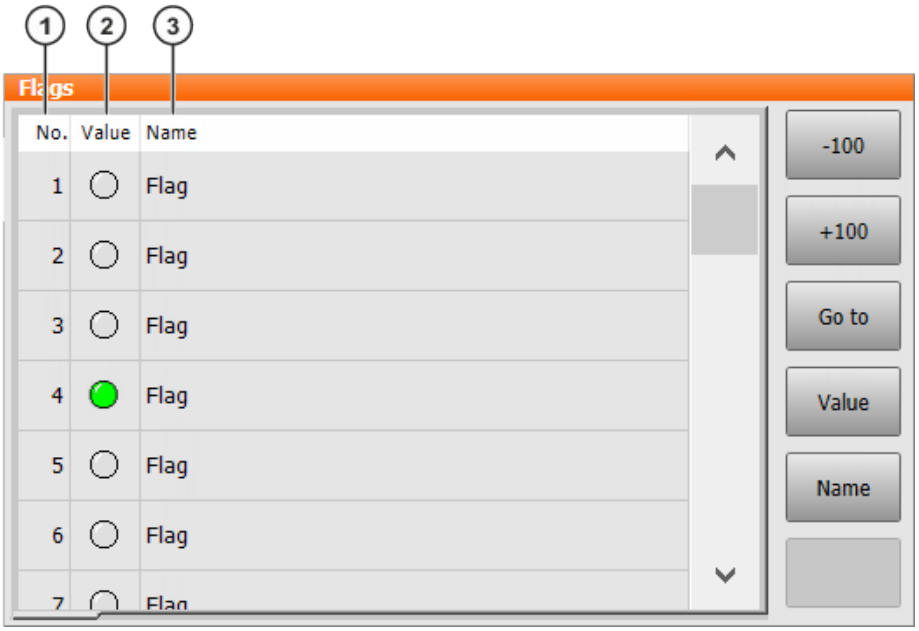


Fig. 4-42: Flags

Item	Description
1	Flag number
2	Value of the flag. The icon is green if a flag is set.
3	Name of the flag

Button	Description
-100	Toggles back 100 flags in the display.
+100	Toggles forward 100 flags in the display.
Go to	The number of the flag being searched for can be entered.
Value	Toggles the selected flag between TRUE and FALSE. Precondition: The enabling switch is pressed. This button is not available in AUT EXT mode.
Name	The name of the selected flag can be modified.
Required user rights for modifications via **Value**, **Name**: Function group **General configuration**	

4.19.10 Displaying counters

Procedure

1. In the main menu, select **Display** > **Variable** > **Counter**. The **Counter** window opens.
2. To display a specific counter:
   - Press the **Go to** button. The **Go to:** box is displayed.
   - Enter the number and confirm with the Enter key.

   The display jumps to the counter with this number.

Description

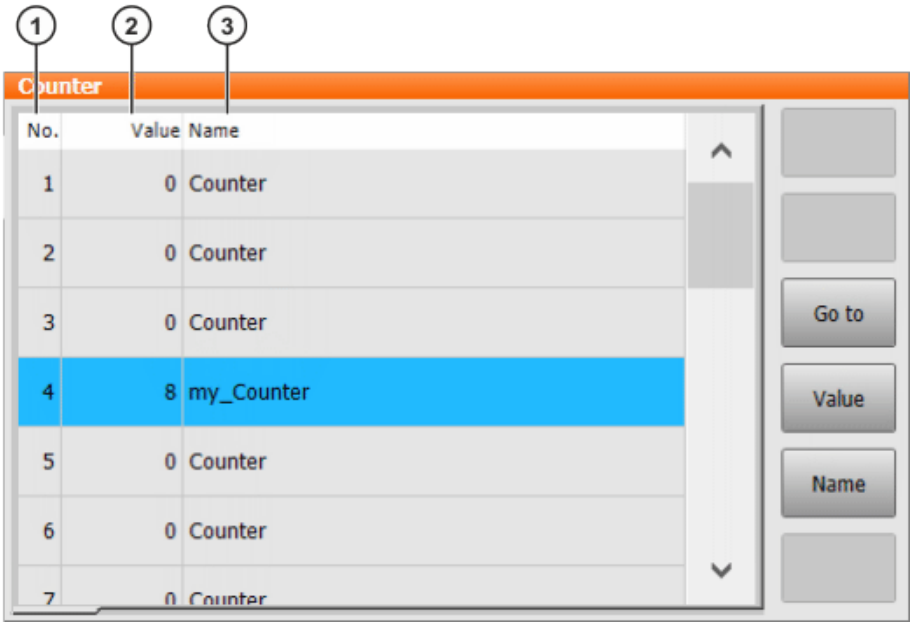


Fig. 4-43: Counter

Item	Description
1	Counter number
4	Value of the counter
5	Name of counter

Button	Description
Go to	The number of the counter being searched for can be entered.
Value	A value can be entered for the selected counter.
Name	The name of the selected counter can be modified.
Required user rights for modifications via **Value**, **Name**: Function group **General configuration**	

4.19.11 Displaying timers

Procedure

1. In the main menu, select **Display** > **Variable** > **Timer**. The **Timer** window is opened.
2. To display a specific timer:
   - Click on the **Go to** button. The **Go to:** box is displayed.
   - Enter the number and confirm with the Enter key.

   The display jumps to the timer with this number.

Description

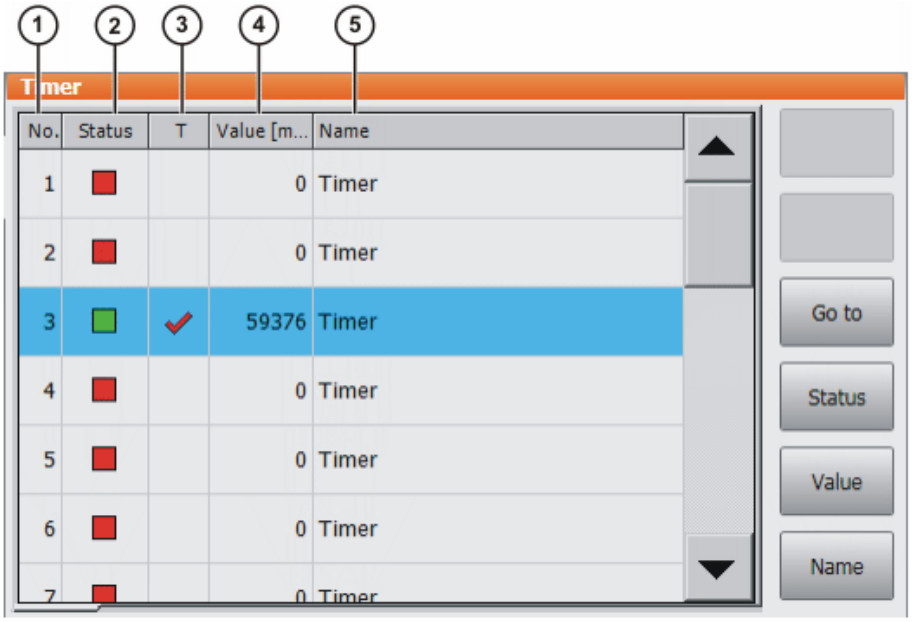


Fig. 4-44: Timer

Item	Description
1	Number of the timer
2	Status of the timer • If the timer is activated, this is indicated in green. • If the timer is deactivated, this is indicated in red.
3	State of the timer • If the value of the timer is > 0, the timer flag is set (red check mark). • If the value of the timer is ≤ 0, no timer flag is set.
4	Value of the timer (unit: ms)
5	Name of timer

Button	Description
Go to	The number of the timer being searched for can be entered.
State	Toggles the selected timer between TRUE and FALSE. Precondition: The enabling switch is pressed.
Value	A value can be entered for the selected timer.
Name	The name of the selected timer can be modified.
Required user rights for modifications via **State**, **Value**, **Name**: Function group **General configuration**	

4.19.12 Displaying information about the robot and robot controller

Procedure

- In the main menu, select **Help** > **Info**.

Description

The information is required, for example, when requesting help from KUKA Customer Support.

The tabs contain the following information:

Tab	Description
Info	• Robot controller type • Robot controller version • User interface version • Kernel system version
Robot	• Robot type and configuration • Service life The operating hours meter is running as long as the drives are switched on. Alternatively, the operating hours can also be displayed via the variable $ROBRUNTIME. • Number of axes • List of external axes • Machine data version

Tab	Description
System	• Controller name • Control PC name • Operating system version • Storage capacities
Options	Additionally installed options and technology packages Also contains the section **Comments**.
Modules	Names and versions of important system files The **Export** button exports the contents of the **Modules** tab to the file C:\KRC\ROBOTER\LOG\FILEVERSIONS.TXT.
EULA	The end user license agreement is displayed here.

In order to be able to execute the function **Export** in **Modules**:

- Required user rights: Function group **Archive to local HDD/SSD**

4.19.13 Displaying/editing robot data

Precondition

- T1 or T2 mode
- No program is selected.
- User rights:
  - To edit data: Function group **Critical configurations**
  - Exception **Save RDC data**: Function group **Archive to USB drives**

Procedure

- In the main menu, select **Start-up** > **Robot data**.

Description

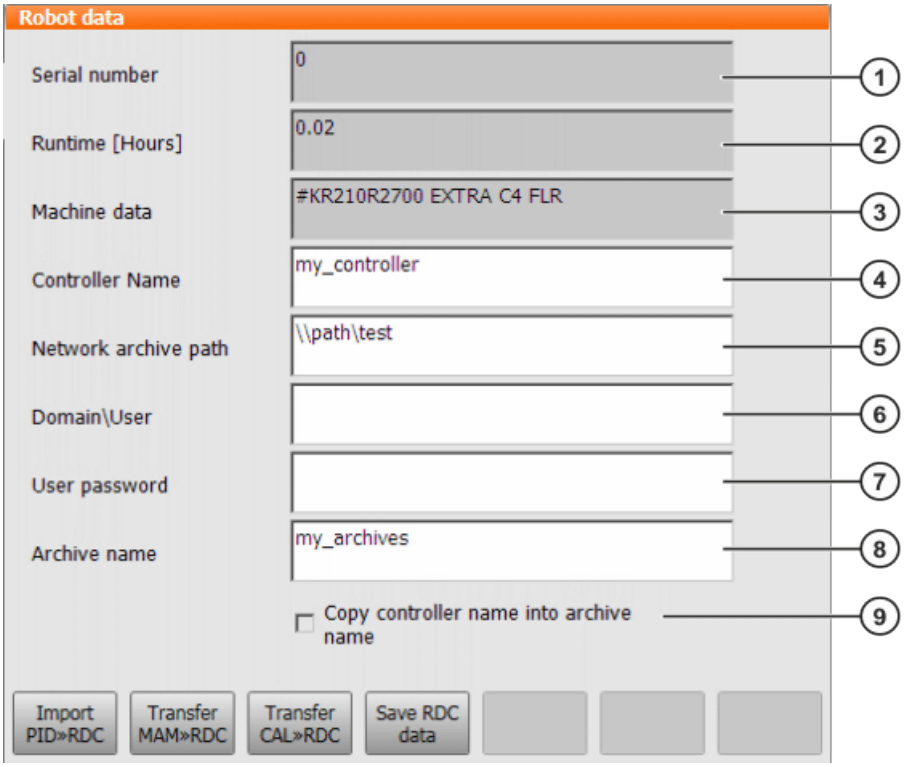


Fig. 4-45: “Robot data” window

Item	Description
1	Serial number
2	Operating hours. The operating hours meter is running as long as the drives are switched on. Alternatively, the operating hours can also be displayed via the variable $ROBRUNTIME.
3	Machine data name
4	Name of the robot controller. The name can be changed.
5	Robot controller data can be archived. The target directory can be defined here. It can be a network directory or a local directory. If a directory is defined here, it is also available for importing/exporting long texts.
6	If archiving to the network requires a user name and password, these can be entered here. It is then no longer necessary to enter them every time for archiving.
7	
8	This box is only displayed if the check box **Incorporate controller name into archive name** is not activated. A name for the archive file can be defined here.
9	• **Check box active**: The controller name is used as the name for the archive file. If no controller name is defined, the name *archive* is used. • **Check box not active**: A separate name can be defined for the archive file.

Buttons

Button	Description
Import PID»RDC	Only relevant for positionally accurate robots: the XML file with the data for the positionally accurate robot can be transferred manually to the RDC. Pressing this button displays the directory structure. The directory containing the file with the current serial number is selected here. The file can be selected and transferred to the RDC.
Transfer MAM»RDC	Only relevant for robots with fixed mastering marks: the MAM file with the robot-specific mastering offset data can be transferred manually to the RDC. Pressing this button displays the directory structure. The directory containing the file with the current serial number is selected here. The file can be selected and transferred to the RDC.
Transfer CAL»RDC	The CAL file with the EMD mastering data can be transferred manually to the RDC. Pressing this button displays the directory structure. The directory containing the file with the current serial number is selected here. The file can be selected and transferred to the RDC.
Save RDC data	The data on the RDC can be backed up temporarily in the directory C:\KRC\Roboter\RDC by pressing this button. **Note**: The directory is deleted when the robot controller is rebooted or data are archived. If the RDC data are to be retained permanently, they must be backed up elsewhere.

4.19.14 Measuring and displaying energy consumption

Description

The energy consumption of the robot and robot controller can be displayed on the smartHMI. The precondition for this is that the robot type used has an energy model.

The smartHMI displays the consumption for the last 60 minutes since the most recent cold start. Furthermore, the user has the option of carrying out measurements himself. There are basically two options:

- Via the smartHMI: In the window **Energy consumption** (>>> *Fig. 4-46*) starting and stopping measurements
- Via KRL

The energy consumption of optional robot controller components and of other controllers is not taken into consideration.

Traces can be made for the consumption values. The predefined configuration Tracedef_KRC_EnergyCalc is available for this. The data can also be transferred to a higher-level controller by means of PROFIenergy. PROFIenergy is a component of KR C4 PROFINET.

Precondition

- The robot type used has an energy model.

- The project was configured in WorkVisual.

  It is not necessary to have a configuration that is specific to the energy model. However, the project must have been transferred from WorkVisual to the controller at least once.
- If the measurement is to be started or stopped via KRL:

  User rights: Function group **General configuration**

If the energy model is not available, the boxes in the **Energy consumption** window are grayed out.

Procedure

Starting and stopping a measurement in the **Energy consumption** window:

1. In the main menu, select **Display** > **Energy consumption**. The **Energy consumption** window opens.
2. If required, activate the check box next to **Refresh**.
3. Press **Start measuring**. A red dot to the right of the top line now indicates that a measurement is in progress.
4. To stop the measurement, press **Stop measuring**. The result is displayed.

Starting and stopping a measurement via KRL:

1. Start the measurement via $ENERGY_MEASURING.ACTIVE = TRUE (possible via the KRL program or variable correction function). The measurement starts.
2. In the main menu, select **Display** > **Energy consumption**. The **Energy consumption** window opens. A red dot to the right of the top line indicates the measurement that is in progress.
3. If required, activate the check box next to **Refresh**.
4. Stop the measurement by means of $ENERGY_MEASURING.ACTIVE = FALSE.

The **Energy consumption** window can also be opened independently of the measurement. The top line always indicates the result of the active or most recent measurement.

Measurement properties

- A measurement that has been started runs until it is stopped. This is not dependent on whether the **Energy consumption** window is open or closed.
- A measurement that has been started via KRL can be stopped via KRL or via the **Stop measuring** button.
- A measurement that has been started by means of **Start measuring** can only be stopped by means of **Stop measuring** as long as the **Energy consumption** window remains open. If an attempt is made to stop the measurement via KRL, the robot controller displays the following message: *Energy measurement cannot currently be stopped*.

  Once the **Energy consumption** window has been closed again, the measurement can also be stopped via KRL. This prevents a measurement started in the **Energy consumption** window from permanently blocking measurements via KRL.
- It is not possible to start a measurement while a measurement is already active. In this case, the robot controller displays the following message: *An energy measurement is already active.*. The active measurement must be stopped first.

“Energy consumption” window

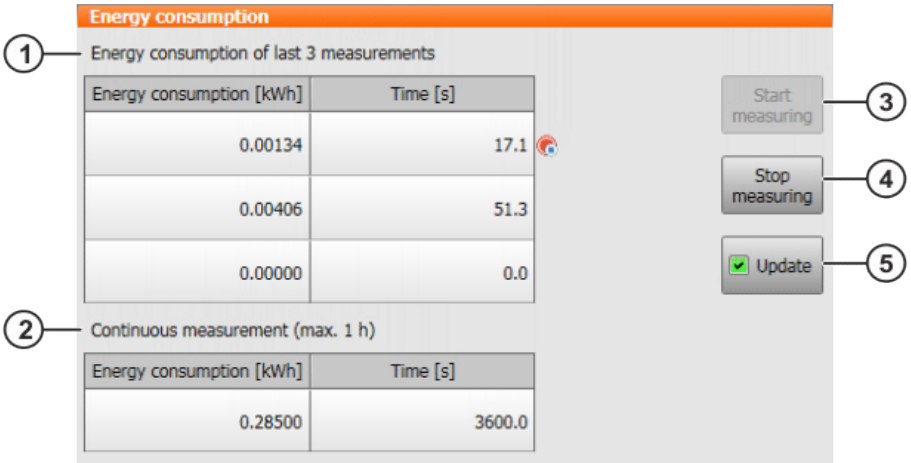


Fig. 4-46: “Energy consumption” window

Item	Description
1	Results of the measurements started by the user The last 3 results are displayed. The most recent result is displayed in the top line. If a measurement is currently active, this is indicated by means of a red dot to the right of the line.
2	Energy consumption for the last 60 minutes since the most recent cold start
3	Starts a measurement. **Start measuring** is not available if a measurement is currently active.
4	Stops an active measurement. How the measurement was started (by means of **Start measuring** or via KRL) is irrelevant.
5	• Check box active: While a measurement is being carried out, the result display is continually refreshed. • Check box not active: While a measurement is being carried out, the most recently refreshed value is displayed. The result is not displayed until the measurement is stopped.

4.19.14.1 $ENERGY_MEASURING: starting and stopping consumption measurement

Description

The user can use $ENERGY_MEASURING to measure the energy consumption of the robot and robot controller. He can start and stop a measurement and read out the results.

The energy consumption of optional robot controller components and of other controllers is not recorded.

Data type: Energy_Measuring_Struc

Energy_Measuring_Struc

```
STRUC Energy_Measuring_Struc BOOL active, locked, REAL
time, energy, lostenergy
```

The REAL components are write-protected.

Element	Description
active	• TRUE: Start measurement • FALSE (default): Stop measurement **Note:** A measurement that has been started via the smartHMI in the **Energy consumption** window cannot be stopped via KRL as long as the **Energy consumption** window remains open. If an attempt is made to stop the measurement via KRL, the robot controller displays the following message: *Energy measurement cannot currently be stopped*. Once the **Energy consumption** window has been closed again, the measurement can also be stopped via KRL. This prevents a measurement started in the **Energy consumption** window from permanently blocking measurements via KRL.
locked	• TRUE: Starting the measurement via the **Energy consumption** window is inhibited. Pressing the **Start measuring**button triggers the following message: *Energy measurement cannot currently be started.* Pressing **Stop measuring** or leaving the **Energy consumption** window automatically sets the variable to FALSE again. • FALSE (default): Starting the measurement via the **Energy consumption** window is possible.
time	Duration of the measurement (s) Also displayed on the smartHMI in the **Energy consumption** window, under **Energy consumption of last 3 measurements** > **Time [s]** (top line)
energy	Energy consumption during measurement period (kWh) Also displayed on the smartHMI in the **Energy consumption** window, under **Energy consumption of last 3 measurements** > **Energy consumption [kWh]** (top line)
lostenergy	The energy produced during the braking process that could not be stored in the intermediate circuit and was converted to heat by way of brake resistance (kWh)

4.19.14.2 $ENERGYMODULE: energy model available for robot?

Description

$ENERGYMODULE indicates whether and to what extent the energy model is available for this robot.

The state of $ENERGYMODULE is relevant for inquiries sent to KUKA Support, for example.

Data type: Energy_Module_Struc

Energy_Module_Struc

```
STRUC Energy_Module_Struc BOOL loaded INT estimated_axes
```

The components are write-protected.

Element	Description
loaded	• TRUE: the energy model is available for this robot. This means that for this robot, consumption can be determined – including external axes. • FALSE: the energy model is not available for this robot.
estimated_axes	Type: INT Bit array denotes how precisely the corresponding energy consumption can be determined for each axis. **Bit 1 … 6**: robot axis A1 ... A6 **Bit 7 … 12**: external axis E1 ... E6 • Bit n = 0: the consumption can be determined precisely for this axis. • Bit n = 1: the consumption can be determined approximately for this axis or is not considered.

Further information about the precision of the relevant axis can be read from $ENERGY_CONFIG_STATE.

Bit n = 1 means that same as $ENERGY_CONFIG_STATE[n] == #APPROXIMATED or #IGNORED.

4.19.14.3 $ENERGY_CONFIG_STATE: status of the energy model for each axis

Description

$ENERGY_CONFIG_STATE[*axis*] = status

Data type: Energy_Config_State_T

$ENERGY_CONFIG_STATE provides the status of the energy model for each axis; in other words, the extent to which the consumption for the axis can be determined.

Energy_Config_State_T

```
ENUM Energy_Config_State_T OK, IRRELEVANT,
APPROXIMATED, IGNORED
```

The component is write-protected.

Element	Description
axis	Axis number Type: INT • **1 … 6**: robot axis A1 ... A6 • **7 … 12**: external axis E1 ... E6
status	Type: ENUM Energy_Config_State_T • #OK: The consumption of this axis can be determined. Or: this axis is not present. • #IRRELEVANT: The axis is irrelevant for the determination of consumption because it is simulated. • #APPROXIMATED: The consumption is approximated for the axis and determined less accurately. • #IGNORED: Only relevant in the simulation case. Since certain model parameters for the simulation are not available for the axis, the consumption cannot be determined.

4.19.14.4 $ENERGY_PERIOD: energy consumption of the last 60 min

Description

$ENERGY_PERIOD indicates the energy consumption of the robot and robot controller during the last 60 min since the most recent cold start.

The energy consumption of optional robot controller components and of other controllers is not recorded.

Data type: Energy_Data_Struc

Energy_Data_Struc

```
STRUC Energy_Data_Struc REAL time, energy, lostenergy
```

The components are write-protected.

Element	Description
time	Duration of the measurement (s) • Maximum: 3,600 s The measurement begins at the last cold start. If the cold start took place more than 60 min ago, the measurement always covers the last 60 min. Also displayed on the smartHMI in the **Energy consumption** window, under **Continuous measurement (max. 1 h)** > **Time [s]**
energy	Energy consumption during measurement period (kWh) Also displayed on the smartHMI in the **Energy consumption** window, under **Continuous measurement (max. 1 h)** > **Energy consumption [kWh]**

Element	Description
lostenergy	The energy produced during the braking process that could not be stored in the intermediate circuit and was converted to heat by way of brake resistance (kWh)

4.19.14.5 $ENERGY_TOTAL: energy consumption since last cold start

Description

$ENERGY_TOTAL indicates the energy consumption of the robot and robot controller since the most recent cold start.

The energy consumption of optional robot controller components and of other controllers is not recorded.

Data type: Energy_Data_Struc

Energy_Data_Struc

```
STRUC Energy_Data_Struc REAL time, energy, lostenergy,
CHAR info[]
```

The components are write-protected.

Element	Description
time	Duration of the measurement (s) Corresponds to the time that has elapsed since the most recent cold start.
energy	Energy consumption during measurement period (kWh)
lostenergy	The energy produced during the braking process that could not be stored in the intermediate circuit and was converted to heat via the brake resistor (kWh)
info[]	For $ENERGY_TOTAL always without a value

4.19.14.6 $ENERGY_INTERIM: energy consumption between time stamps

Description

$ENERGY_INTERIM displays the energy consumption of the robot and robot controller in the time between the two most recent calls of the INTERIMENERGY() function. The initial value in the case of cold start is 0.

The energy consumption of optional robot controller components and of other controllers is not recorded.

Data type: Energy_Data_Struc

Energy_Data_Struc

```
STRUC Energy_Data_Struc REAL time, energy, lostenergy,
CHAR info[]
```

The components are write-protected. Exception: `info[]` can be set using the INTERIMENERGY() function.

Element	Description
time	Duration of the measurement (s) Corresponds to the time that has elapsed between the two most recent calls of the INTERIMENERGY() function.
energy	Energy consumption during measurement period (kWh)

Element	Description
lostenergy	The energy produced during the braking process that could not be stored in the intermediate circuit and was converted to heat by way of brake resistance (kWh)
info[]	The string that was transferred during the most recent call of the INTERIMENERGY() function

4.19.14.7 INTERIMENERGY(): Setting a time stamp for the measurement of energy consumption

Description

The INTERIMENERGY function sets up a time stamp. Energy consumption values are written to $ENERGY_INTERIM using the stamp (= function call). When $ENERGY_INTERIM is read out, you obtain the values incurred between the second-to-last and final stamp.

Values from previous stamp intervals are overwritten. If these values are to be saved, the user must ensure that they are written to special-purpose variables prior to every new function call.

Syntax

result = `INTERIMENERGY` (*string*)

Explanation of the syntax

Element	Description
result	Type: INT Variable for the return value • 1: The function was called successfully. • <>1: Bug
string	Type: CHAR Type of transfer: IN parameter Any string. Serves as additional info on the respective function call.

Example

KRL program:

The function calls have been marked as “nr_1”, “nr_2”, etc. using the parameter *string* in this case.

```
1 ...
2 my_result = interimenergy ("nr_1")
3 ...
4 my_result = interimenergy ("nr_2")
5 ...
6 my_result = interimenergy ("nr_3")
7 ...
```

Reading out $ENERGY_INTERIM:

When $ENERGY_INTERIM is read out, you obtain the values incurred between the second-to-last and final stamp.

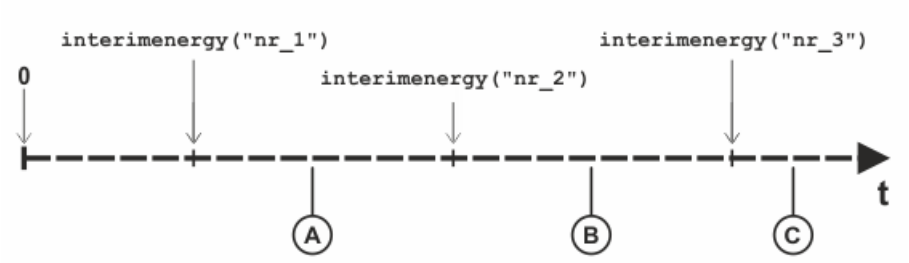


Fig. 4-47: Chronological sequence

Item	Description
0	Cold start
Reading out $ENERGY_INTERIM:	
A	Reading out between "nr_1" and "nr_2" displays the following consumption: • From cold start up to "nr_1"
B	Reading out between "nr_2" and "nr_3" displays the following consumption: • From "nr_1" to "nr_2"
C	Reading out after "nr_3" (and up to a potential "nr_4") displays the following consumption: • From "nr_2" to "nr_3"

$ENERGY_INTERIM also displays other values: Duration of measurement period, string from the most recently completed function call , unstored energy.

4.20 Displaying the battery state

Description

If the voltage is switched off (i.e. via the main switch) or in the event of power failure, the robot controller is backed up by a battery and is shut down in a controlled manner (without loss of data). The battery charge can be displayed for the user. The user can also transfer it to the PLC.

The battery charge is displayed by means of the system variable $ACCU_STATE.

The state can only be displayed and not modified.

The charging current characteristic is monitored every time the robot controller is booted. An additional battery test is carried out cyclically. The state indicated by $ACCU_STATE is derived from the information regarding the charging current and the battery test.

States

The following tables indicate the possible states of $ACCU_STATE.

The user must configure the info to the PLC himself.

Information about exchanging the battery can be found in the operating instructions for the robot controller.

#CHARGE_OK
Meaning: The charging current dropped as required after booting and/or the battery tested positive in the battery test.
Action required by the user: Do not exchange the battery.
Info to PLC: Supply voltage disconnection OK.
Message: No message.

#CHARGE_OK_LOW
Meaning: The charging current dropped as required after booting and/or the battery tested positive in the battery test. The battery is not fully charged, however, after the maximum charging time.
Action required by the user: Exchange the battery.
Info to PLC: Supply voltage disconnection OK.
Message: *Battery warning - full charge not possible*

#CHARGE_UNKNOWN
Meaning: The battery is being charged. Or the battery has not yet been checked since the controller was booted. Or the charging current has not yet dropped sufficiently.
Action required by the user: Do not exchange the battery.
Info to PLC: Supply voltage disconnection can cause errors in Hibernate mode.
Message: No message

#CHARGE_TEST_NOK
Meaning: The result of the battery test was negative.
Action required by the user: Exchange the battery.
Info to PLC: Supply voltage disconnection can cause errors in Hibernate mode.
Message: *Battery defective - load test failed*

#CHARGE_NOK
Meaning: No battery test possible. The battery is not fully charged after the maximum charging time.
Action required by the user: Exchange the battery.
Info to PLC: Supply voltage disconnection can cause errors in the case of a warm start.
Message: *Battery defective - reliable backup cannot be assured*

#CHARGE_OFF
Meaning: There is no battery present or the battery is defective.
Action required by the user: Exchange the battery.
Info to PLC: Supply voltage disconnection can cause errors in the case of a warm start.
Message: *Battery defective - backup not possible*

5 Start-up and recommissioning

5.1 Protecting the Windows system

KUKA recommends the following measures for protecting the Windows system:

Measure	Description
Use **KUKA.NonAdmin**	Add-on functionality for protection of the Windows system against unwanted access
Use **KUKA.CPC** or **KUKA.Ikarus**	KUKA.CPC: Add-on functionality for protection of the Windows system against unauthorized access, protection against malicious software
	KUKA.Ikarus: Add-on functionality for protection against malicious software
Change the default password	(>>> *5.3 "Changing the password for a Windows system" Page 129*)

There is separate documentation for the software products **KUKA.NonAdmin**, **KUKA.CPC** and **KUKA.Ikarus**.

5.2 Switching on the robot controller with the kinematic system for the first time

Description

When the robot controller with connected kinematic system is switched on for the first time, it is imperative for the following procedure to be observed.

This procedure is not a substitute for a complete initial start-up of the robot system!

DANGER

In the absence of functional safety equipment and safeguards, the robot system can cause personal injury or material damage. If safety equipment or safeguards are dismantled or deactivated, the robot system may not be operated.

WARNING

Use of Start-up mode disables all external safeguards. The service personnel are responsible for ensuring that there is no-one in or near the danger zone of the manipulator as long as the safeguards are disabled. Failure to observe this precaution may result in death, injuries or damage to property.

Procedure

1. Switch on the robot controller.
2. Select T1 mode.
3. Turn the keyswitch (>>> *Fig. 5-1*) to the **Operation** position (symbol: open padlock).

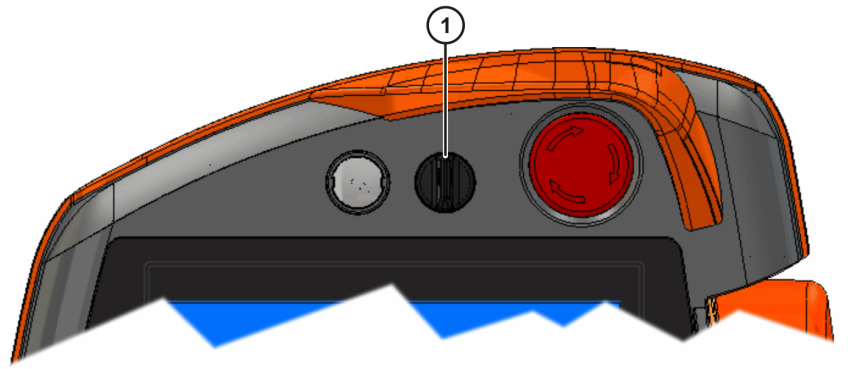


Fig. 5-1: Keyswitch

Item	Description
1	Keyswitch

4. The following query is displayed: **Inconsistency between RDC memory and controller. What has been exchanged?** (>>> *Fig. 5-2*)
   Click on the **Robot** button to confirm.

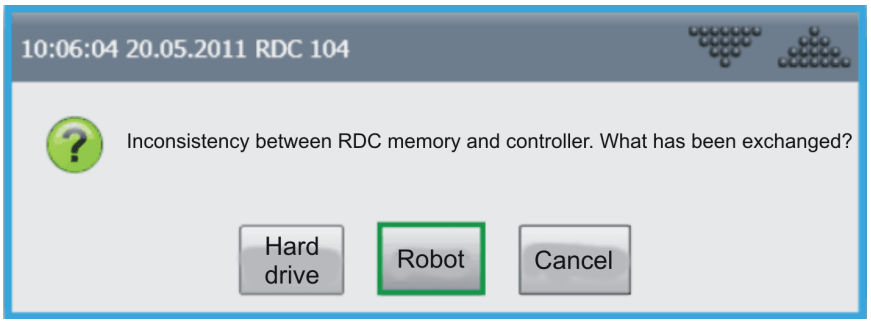


Fig. 5-2: Inconsistency between RDC memory and controller

5. The following message is displayed: **Network configuration restored**.
   Click on the **OK** button to confirm.
6. Confirm in the message window all messages that can be acknowledged by clicking on the **Confirm all** button. (>>> *Fig. 5-3*)

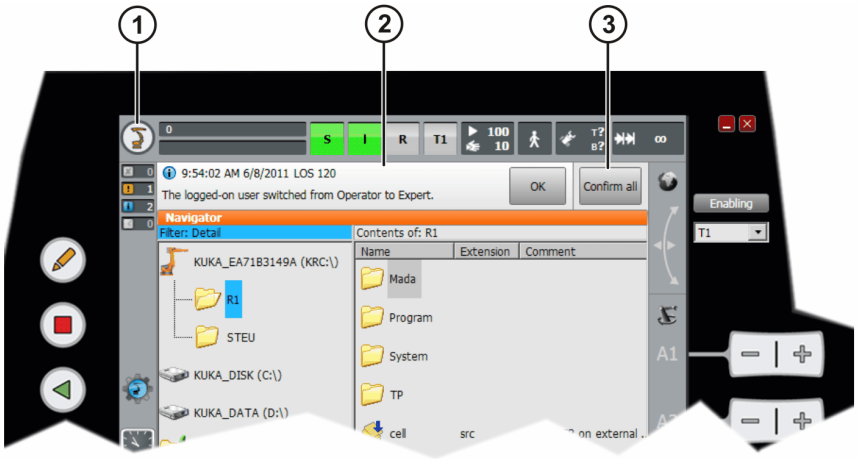


Fig. 5-3: Message window

Item	Description
1	Main menu key
2	Message window
3	**Confirm all** button

7. The following messages cannot be acknowledged yet:
   - **KSS15068** *Safety configuration checksum error.*
   - **KSS12017** *Operator safety not acknowledged*
   - **KSS00404** *Safety stop*

   To allow them to be acknowledged, the safety configuration of the robot (RDC) and the robot controller must be synchronized:

   Press the main menu key and select the menu sequence **Configuration** > **User group**. Then press **Login...**.
8. Select the user group **Safety Recovery**. If KUKA.SafeOperation is used, select the user group **Safety Maintenance**.
9. Enter the password (default: **kuka**) and confirm with **Login**.
10. Acknowledge the messages **KSS15068** and **KSS00404**:
    - Press the main menu key and select the menu sequence **Configuration** > **Safety configuration**.
    - An overview of possible causes of errors is displayed in the window **Troubleshooting wizard**. (>>> *Fig. 5-4*)

      Select the following entry in the list: *The robot or the RDC memory is being started up for the first time.*

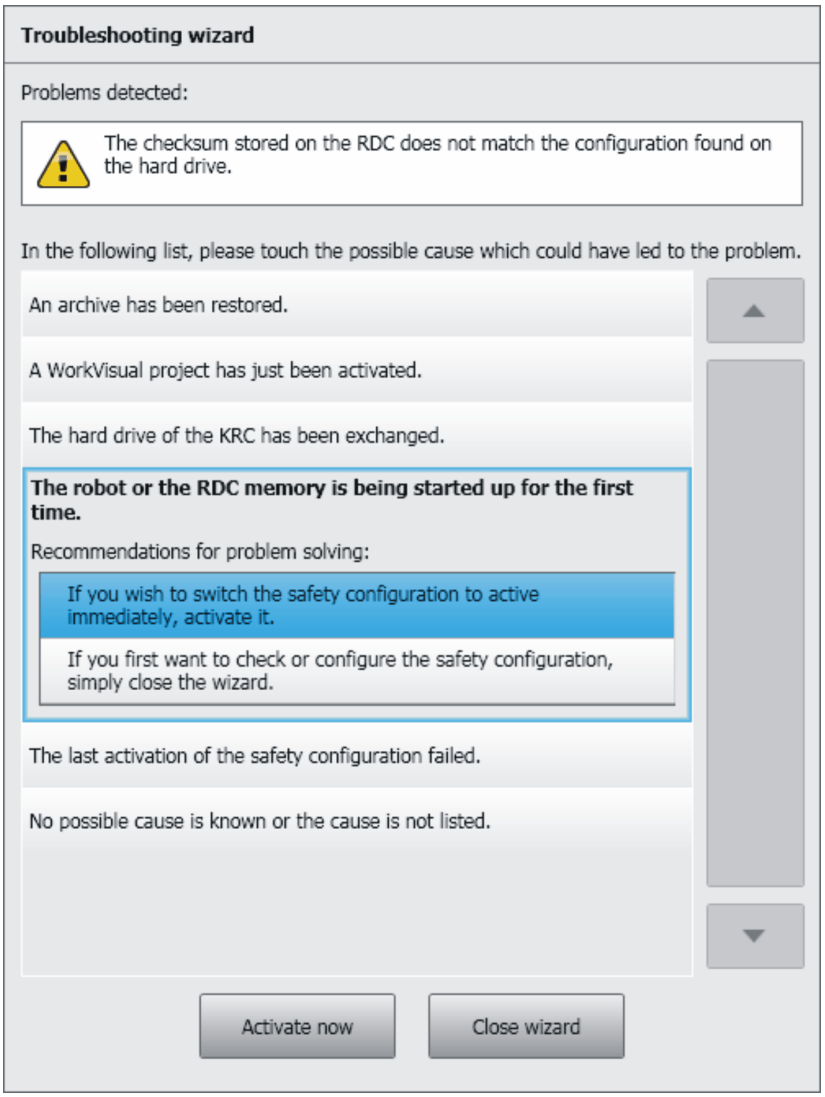


Fig. 5-4: Troubleshooting wizard

- Recommendations for remedial action are shown under the selected entry.

  Select the following entry here: *If you wish to switch the safety configuration to active immediately, activate it.*
- Press the **Activate now** button. The **Safety configuration** window opens. The **Common** tab is displayed.

  Depending on the version of the system software, the **Common** tab may vary from the illustration (>>> *Fig. 5-5*).

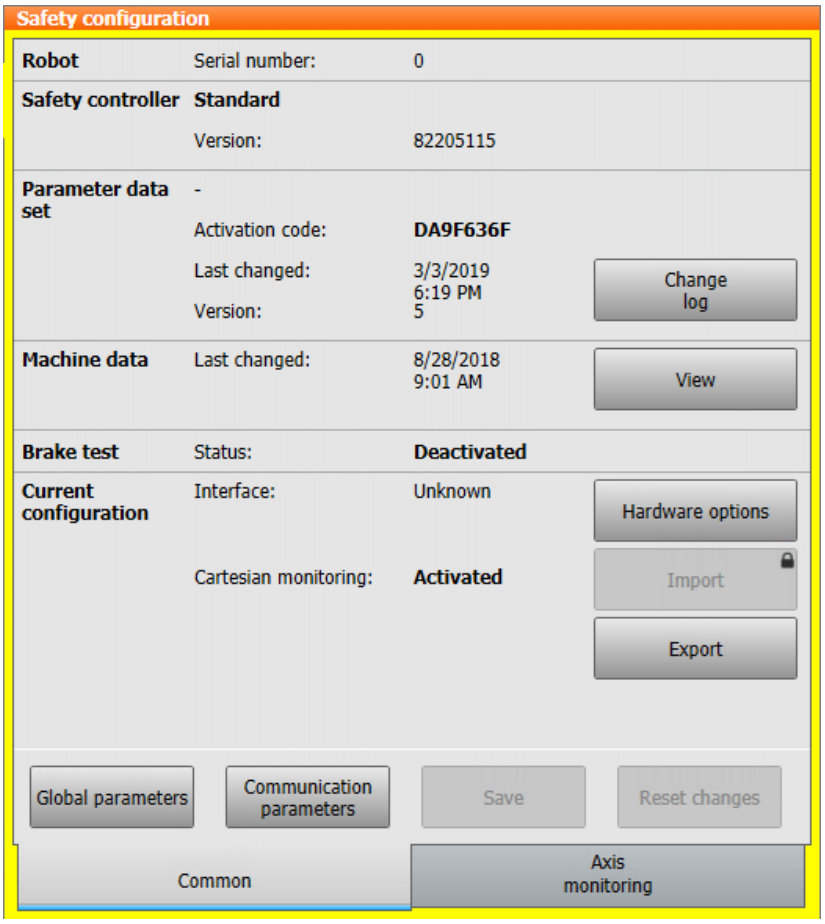


Fig. 5-5: “General” tab

- Close the **Safety configuration** window using the **Close** icon .
- The following message is displayed: **The changes were saved successfully.**

  Click on the **OK** button to confirm.

11. Bring the robot into an operational state (2 cases are distinguished here):
    - If the robot already has a fully configured and wired safety interface: acknowledge message 12017.
    - Move the robot in Start-up mode. In the case of a safety interface based on a field bus (ProfiSAFE/CIP Safety/FSoE), it is only possible to switch to Start-up mode if there is no connection to the field bus. Press the main menu key and select the menu sequence **Start-up** > **Service** > **Start-up mode**.

The KR C4 is ready for operation. A complete initial start-up can be carried out.

5.3 Changing the password for a Windows system

Description

When starting Windows, the user is automatically logged on with the following data:

- User name: **KukaUser** (Has administrator rights by default.)
- Password: **68kuka1secpw59**

The password can be changed using the procedure described here. There are no restrictions for the password, e.g. in terms of length or complexity.

NOTICE
If the changed password is lost, access to the Windows system is no longer possible – not even via KUKA Deutschland GmbH!

Precondition

- Administrator rights

Procedure

1. Open the Windows Start menu and select **Run…**.
2. Enter the `cmd` command in the **Open** box and confirm with the Enter key. The command window opens.
3. Enter the following command:
   `c:\krc\util\krcuserpw\changepwd.exe /u=kukauser /op=OLD_PW /p=NEW_PW /cp`
   Here, enter the current password instead of `OLD_PW` and the desired new password instead of `NEW_PW`.
4. Confirm by pressing the Enter key.
   The password is changed. There is no confirmation message. The change is effective immediately.

Parameters

Parameter	Description
/u= …	User name
/op= …	Current password Upper and lower case are taken into consideration.
/p= …	New password Upper and lower case are taken into consideration.
/cp	“Change password” command

Log file

The change is logged in:

- C:\KRC\ROBOTER\LOG_ChangePwd.log

The errors are also indicated here in plain text. Password changes via WorkVisual are logged here, too.

5.3.1 Changing the password: return values

The command line supplies return values which can be checked in MS DOS via ERRORLEVELS. The return value is “0” if the password has been successfully changed.

Return values in the event of errors (positive integers):

Code		Description
ERR_OPENFILE	0x0001	Error opening the KEC file
ERR_ARGMISMATCH	0x0002	No /u or /s parameter has been specified.
ERR_ADDUSERTOGROUP	0x0004	The user cannot be added to the group.

Code		Description
ERR_USERNAME_EMPTY	0x0008	No user name was specified for “Change password”.
ERR_PASSWORD_EMPTY	0x0010	No new password was specified for “Change password”.
ERR_READINGUSER_KUKA-CONFIG	0x0020	The VxWorks user cannot be read from kuka.config* for “Change password”.
ERR_READINGPASSWORD_KU-KACONFIG	0x0040	The VxWorks password cannot be read from kuka.config* for “Change password”.
ERR_EXTRACTINGIN-FO_STARTKRCKEC	0x0080	The following data cannot be read from StartKrc.kec in the path C:\KRC for “Change password”: • User name, domain and password of the KEC user
ERR_WRITETARGETPASS-WORD	0x0100	The changed password cannot be written to kuka.config* for “Change password”.
ERR_OPENSOURCEKECFILE	0x0200	The contents of the existing KEC file cannot be read when creating the KEC file.
ERR_SETSTARTUSER	0x0400	Auto logon user cannot be set for “Change password”.
ERR_GETSTARTUSER	0x0800	The start user cannot be determined from the StartUser.bin file for “Change password”. StartUser.bin path: • System Software 8.3 to 8.5: C:\Windows \System32 • System Software 8.6 or higher: C:\Windows\SysWOW64
ERR_COMMANDINPUTFILEAC-CESSERROR	0x1000	There is no response file with the options for the password change or it cannot be accessed.
ERR_WRONG_NONADMIN-PACKAGE	0x2000	The installed NonAdmin version is not suitable for the password change.
ERR_WRONG_CPCPACKAGE	0x4000	The installed CPC version is not suitable for the password change.

*Path of the kuka.config file: C:\KRC\ROBOTER\Config\System\Common \VxWin

Return values in the event of errors (negative numbers):

Code		Description
ERROR_ACCESS_DENIED	-5	The user does not have access rights.
ERROR_INVALID_PASSWORD	-86	The user has entered an invalid password.
ERROR_INVALID_PARAMETER	-87	Invalid parameter
NERR_InvalidComputer	-2351	Invalid computer name
NERR_NotPrimary	-2226	The operation is only allowed on the primary domain controller.
NERR_UserNotFound	-2221	The user name could not be found.
NERR_PasswordTooShort	-2245	Password too short

The return values come from the NetUserchangePassword method. For reasons of completeness, return values are also specified here which are not relevant for the robot controller, e.g. NERR_PasswordTooShort. For the robot controller, there are NO restrictions for the password, e.g. in terms of length or complexity.

5.4 Start-up wizard

Description

Start-up can be carried out using the Start-up wizard. This guides the user through the basic start-up steps.

Precondition

- No program is selected.
- Operating mode T1

Procedure

- Select **Start-up** > **Start-up wizard** in the main menu.

5.5 Modifying the machine data configuration

Description

The machine data of the industrial robot are configured in WorkVisual. If necessary, the machine data configuration can be modified.

In the case of a standard 6-axis robot without external axes, the machine data of the following parameter groups can be edited, for example:

- **Software limit switches** (all axes)
- **Controller parameters** (all axes)
- **Warm-up parameters**

Other machine data, e.g. the cell parameter “Root point” are only displayed and cannot be modified.

Information about the individual machine data can be found in the documentation **Configuration of Kinematic Systems**.

Information about the warm-up parameters can be found in this documentation. (>>> *6.16.2 "Configuring warm-up" Page 234*)

Precondition

- User rights: Function group **General configuration**
- T1 or T2 mode
- No program is selected.

Procedure

1. In the main menu, select **Configuration** > **Machine configuration**.
2. Modify the machine data as required and press **Save**.
3. Answer the request for confirmation with **Yes**. The submit interpreters are automatically deselected while the data are saved and then reselected.

5.6 Defining hardware options

Precondition

- User group Safety Maintenance Technician or higher
- T1 or T2 mode

Procedure

1. In the main menu, select **Configuration** > **Safety configuration**.
2. Press **Hardware options**.
3. Modify hardware options and press **Save**.

WARNING

Following modifications to the safety configuration, the values for the safe axis monitoring functions must be checked.
(>>> *6.5 "Checking the values for the safe axis monitoring functions" Page 214*)

Description

Parameter	Description
Customer interface	Select here which interface is used: • **Automatic** • **SIB with operating mode output** **Note:** While this box is displayed for certain controller variants, it is grayed out and has no effect. Affected variants: compact, smallsize, smallsize-2. The box still has no effect for these variants, even if the setting has been modified! This may be the case, for example, if a different controller variant was previously selected.
Input signal for peripheral contactor (US2)	**Deactivated**: the peripheral contactor is not used. (Default) **By external PLC**: the peripheral contactor is switched by an external PLC via input “US2”. **By KRC**: the peripheral contactor is switched if the following conditions are met: • No operator safety message is active. • No E2/E7 message is active. (Only relevant for VSS.) • Drives are switched on. • The motion enable signal is present. **Notes:** • For robot controllers with peripheral contactors and the “UL” option, the setting **By KRC** must be selected. • For robot controllers with no peripheral contactors, this box is grayed out and has no effect. • $CRIT_PERI_ACK_REQ can influence the behavior of the US2 peripheral contactor. (>>> *"Peripheral contactor (US2)" Page 139*) The system variable $US2_VOLTAGE_ON indicates the status of the peripheral voltage US2: • TRUE: voltage is switched on. • FALSE: voltage is switched off.
Operator safety acknowledgement	If the “Operator Safety” signal is lost and set again in Automatic mode, it must be acknowledged before operation can be continued. • **By acknowledgement button**: Acknowledgement is given, for example, by an acknowledgement button (situated outside the cell). Acknowledgement is communicated to the safety controller. The safety controller re-enables automatic operation only after acknowledgement. • **External unit**: Acknowledgement is given by the system PLC.

5.7 Preventing an unexpected start of the peripheral devices

5.7.1 Behavior with/without additional enabling

Default behavior (= without additional enabling)

If there is a peripheral device (e.g. a milling tool) controlled via the PLC in the cell, it is possible for the device to be started up in T1 and T2 mode just by pressing the enabling switch. In such a case, the robot or an external axis cannot yet start up.

It is possible to prevent such an unexpected start-up of the peripheral device by configuring an additional enabling signal.

WARNING

The described start-up through the simple use of the enabling switch is not permissible according to applicable standards if as a result there is a possibility of the peripheral device causing damage to property or injury.

- According to applicable standards, the device may only start up after additional operator action.
- If no additional operator action is carried out, property damage, injuries or death to persons may result.
- According to applicable standards, there are several options for what the specific additional operator action may look like.

  With the “additional enabling” described here, KUKA offers a standard-compliant option.

Behavior with additional enabling

If additional enabling is configured, the behavior is as follows:

- If a peripheral device is present, the robot controller displays the message M_3303 upon pressing the enabling switch:

  Acknowledge or press the Start key to enable critical peripheral equipment.
- The peripheral device starts up only once the message has been acknowledged or a start key (“Start forwards” or “Start backwards”) has been pressed.

While the message is active, the robot can be moved in T1 using the jog keys without the peripheral device starting up.

5.7.2 Configuring additional enabling

Precondition

$machine.dat must be edited for the configuration. This can be done on the robot controller or in WorkVisual.

Precondition if the $machine.dat file is to be edited on the robot controller:

- User rights: function group **Critical KRL program changes**

Procedure

Make the settings in the $machine.dat file:

1. Open the $machine.dat file under KRC:\STEU\Mada.
2. Allocate SIGNAL $CRIT_PERI_ACK_REQ either to $IN[1025] or to another input.

- $IN[1025]: the additional enabling is permanently active since $IN[1025] is permanently TRUE
- Other input: the additional enabling can be set to active or inactive via this input

3. Allocate SIGNAL $CRIT_PERI_ACK to an output.

> This step is only necessary in the following cases:
> – If **Input signal for peripheral contactor (US2)** = **By external PLC** is set on the robot controller.
> – Or if the PLC switches the peripheral equipment via another mechanism (i.e. without using US2).

$CRIT_PERI_ACK informs the PLC via this output whether there is an enabling signal for starting up the peripheral device.

4. Close the $machine.dat file. Respond to the request for confirmation asking whether the changes should be saved by pressing **Yes**.

Configuring the PLC:

- Configure the PLC in such a way that it can correctly evaluate the output of the robot controller to which SIGNAL $CRIT_PERI_ACK is allocated.

RoboTeam

> - The configuration of $CRIT_PERI_ACK_REQ must be same for all participants in a RoboTeam.
> - And: $CRIT_PERI_ACK_REQ must simultaneously be TRUE for all participants in a RoboTeam.

If $CRIT_PERI_ACK_REQ is TRUE for all participants, the following applies:

- The M_3303 message is simultaneously generated by or removed from all controllers in the RoboTeam.
- If the user acknowledges the M_3303 message on one controller, it is automatically acknowledged for all other participants.
- The signal output $CRIT_PERI_ACK (if used) behaves identically with all participants.

  For the different participants, $CRIT_PERI_ACK can be allocated to different outputs.

5.7.3 SIGNAL $CRIT_PERI_ACK

Description

If addtional enabling is to be used for starting up peripheral devices, SIGNAL $CRIT_PERI_ACK must be configured in the following cases:

- If **Input signal for peripheral contactor (US2)** = **By external PLC** is set on the robot controller.
- Or if the PLC switches the peripheral equipment via another mechanism (i.e. without using US2).

Properties:

- SIGNAL $CRIT_PERI_ACK is implemented as a digital output and can be allocated to any $OUT[].
- $CRIT_PERI_ACK informs the PLC via the assigned output whether there is an enabling signal for starting up the peripheral device.

- $CRIT_PERI_ACK is write-protected.

Default in KRC:\STEU\Mada\$machine.dat:

```
SIGNAL $CRIT_PERI_ACK FALSE
```

Example: SIGNAL $CRIT_PERI_ACK has been allocated to $OUT[66]:

```
SIGNAL $CRIT_PERI_ACK $OUT[66]
```

Status

$CRIT_PERI_ACK status:

Status	Description
FALSE (= default)	There is no enabling signal for starting up the peripheral device.
TRUE	There is an enabling signal.

5.7.4 SIGNAL $CRIT_PERI_ACK_REQ

Description

- SIGNAL $CRIT_PERI_ACK_REQ is implemented as a digital input and can be allocated to any $IN[].
- $CRIT_PERI_ACK_REQ sets the “additional enabling” function to active or inactive.
- $CRIT_PERI_ACK_REQ is write-protected.

Default in KRC:\STEU\Mada\$machine.dat:

```
SIGNAL $CRIT_PERI_ACK_REQ $IN[1026]
```

In the default setting, the “additional enabling” function is permanently inactive since $IN[1026] is permanently FALSE.

Example: SIGNAL $CRIT_PERI_ACK_REQ has been allocated to $IN[66]:

```
SIGNAL $CRIT_PERI_ACK_REQ $IN[66]
```

If it is desired that the “additional enabling” is permanently active, SIGNAL $CRIT_PERI_ACK_REQ must be allocated to $IN[1025]. $IN[1025] is permanently TRUE.

Status

$CRIT_PERI_ACK_REQ status:

Status	Description
FALSE (= default)	The “additional enabling” function is inactive. This means that switching the US2 is only possible by pressing the enabling switch. Status of $CRIT_PERI_ACK if $CRIT_PERI_ACK_REQ == FALSE: • **T1/T2**: FALSE • KSS: **AUT/AUT EXT**: TRUE VSS: **EXT**: TRUE • **Invalid operating mode**: FALSE
TRUE	The “additional enabling” function is active. Status of $CRIT_PERI_ACK if $CRIT_PERI_ACK_REQ == TRUE: • **T1/T2**: if the enabling switch has been pressed and the safety controller gives the motion enable signal, the M_3303 message is generated. $CRIT_PERI_ACK remains FALSE. After acknowledging the M_3303 message or pressing “Start”, $CRIT_PERI_ACK goes to TRUE. If the safety controller gives the motion enable signal, $CRIT_PERI_ACK becomes FALSE again. **Note:** This behavior also applies for start-up mode. • KSS: **AUT/AUT EXT**: TRUE VSS: **EXT**: TRUE • **Invalid operating mode**: FALSE

The M_3303 message is only generated in the combination of $CRIT_PERI_ACK_REQ == TRUE with T1 or T2 mode. It is never generated in any other combinations. If the message is active while switching to another combination, the robot controller removes the message.

Peri enabled (PE)

If $CRIT_PERI_ACK_REQ == TRUE, the “Peri enabled” (PE) system output goes to TRUE under the following conditions:

- $CRIT_PERI_ACK == TRUE
- And: all further conditions for switching on “Peri enabled” (PE) are met.

Peripheral contactor (US2)

"Input signal for peripheral contactor (US2)" = …	Behavior if $CRIT_PERI_ACK_REQ == TRUE
Deactivated	The US2 peripheral contactor is inactive. In this case, $CRIT_PERI_ACK_REQ and US2 have no effect on one another. Since $CRIT_PERI_ACK_REQ == TRUE, the M_3303 message is generated if the preconditions are met. $CRIT_PERI_ACK can be used by the PLC as a criterion to switch the peripheral equipment via another mechanism (i.e. without US2).
By external PLC	The US2 peripheral contactor switches to ON if the PLC sets the “US2” input to TRUE. $CRIT_PERI_ACK can be used by the PLC as a criterion to switch the US2 peripheral contactor.
By KRC	The US2 peripheral contactor switches to ON under the following conditions: • $CRIT_PERI_ACK == TRUE • And: all further conditions for switching on the US2 peripheral contactor are met.

5.8 Changing the safety ID of the PROFINET device

Description

If multiple KUKA robot controllers are operated with a single PROFIsafe master PLC, each PROFINET device must have a unique safety ID. The default ID is always 7.

Precondition

- User group “Safety recovery” or higher
- T1 or T2 mode

If a safety option is installed on the robot controller, e.g. KUKA.SafeOperation, different preconditions may apply. Information can be found in the documentation of the safety options.

Procedure

1. In the main menu, select **Configuration** > **Safety configuration**.
2. Press **Communication parameters**.
3. In the column **New safety ID**, press the ID to be modified and change the ID.
4. Press **Apply safety IDs**.
5. A request for confirmation is displayed, asking if the change should be saved. Confirm the request with **Yes**.
6. A message is displayed, indicating that the change has been saved. Confirm the message with **OK**.

This procedure can only be used to save changes to the safety ID. If other unsaved changes have been made elsewhere in the safety configuration, these are not saved here.

If an attempt is now made to close the safety configuration, a query is generated asking whether you wish to reject the changes or cancel the action. To save the changes, proceed as follows:

1. Cancel the action.
2. In the safety configuration, press **Save**. (If the **Save** button is not available, first go back a level by pressing **Back**.)
3. A request for confirmation is displayed, asking if all the changes should be saved. Confirm the request with **Yes**.
4. A message is displayed, indicating that the change has been saved. Confirm the message with **OK**.

   All changes in the safety configuration are saved.

WARNING

Following modifications to the safety configuration, the values for the safe axis monitoring functions must be checked.
(>>> *6.5 "Checking the values for the safe axis monitoring functions" Page 214*)

5.9 Switching safe Cartesian monitoring on/off

Description

Safe Cartesian monitoring refers to both KUKA kinematic systems as well as customer kinematics (CKs).

As a rule, the settings for safe Cartesian monitoring must not be changed on the controller. The reason is that WorkVisual projects almost always have only one possible setting: either active or inactive. They can only be transferred to the robot controller if the setting is correct.

The setting for safe Cartesian monitoring can, however, be changed in the following cases:

- If simulated axes are used, active monitoring can be deactivated.

  This is the only case in which the user can decide whether to operate the kinematic system with or without monitoring.
- If the robot controller indicates via message that monitoring is not possible.

  If simulated axes are used and monitoring is active in WorkVisual, it can only first be determined during the test on the robot controller whether the kinematic system can be moved. (It is not possible to determine this before in WorkVisual.) If the kinematic system cannot be moved, a message is generated which indicates that Cartesian monitoring is not possible.

WARNING

To operate a kinematic system for which safe Cartesian monitoring is not possible, this monitoring can be deactivated. Before deactivation, a risk assessment must be carried out. Failure to carry this out may result in death to persons or severe injuries.

The **Cartesian monitoring:** check box refers to all Cartesian safety functions, including safe Cartesian monitoring in T1 mode. The check box is always present, irrespective of whether a safety option is used or not.
There is always a basic, non-safe limitation of the speed in T1 mode to <= 250 mm/s as well as a safe monitoring of the axis-specific speed. They are not influenced by the setting **Cartesian monitoring:**.

Precondition

- User group “Safety maintenance” or higher
- T1 or T2 mode
- No program is selected.

Procedure

1. In the main menu, select **Configuration** > **Safety configuration**.
   The **Safety configuration** window opens.
2. On the **Common** tab, press **Global parameters**.
3. Remove the check mark at **Cartesian monitoring:** and press **Save**.
4. Answer the request for confirmation with **Yes**. The controller is reconfigured.
5. Once the reconfiguration has been completed, the following message is displayed: *The changes were saved successfully.*
   Confirm the message with **OK**.

5.10 Jogging the robot without a higher-level safety controller

Description

To jog the robot without a higher-level safety controller, Start-up mode must first be activated. The robot can then be jogged in T1 mode.

DANGER

External safeguards are disabled in Start-up mode. Observe the safety instructions relating to Start-up mode.
(>>> *3.8.4.2 "Start-up mode" Page 44*)

The robot controller automatically deactivates Start-up mode in the following cases:

- If no operator action has been carried out within 30 min of activation.
- If the smartPAD is switched to passive mode or disconnected from the robot controller.
- If the Ethernet safety interface is used: when a connection to a higher-level safety controller is established.
  (If a discrete safety interface is used, Start-up mode is not dependent on the inputs at the discrete safety interfaces.)

Effect

When the Start-up mode is activated, all outputs are automatically set to the state “logic zero”.

If the robot controller has a peripheral contactor (US2), and if the safety configuration specifies for this to switch in accordance with the motion enable, then the same also applies in Start-up mode. This means that if mo-

tion enable is present, the US2 voltage is switched on – even in Start-up mode.

NOTICE
The maximum number of switching cycles of the peripheral contactors is 175 per day.

In Start-up mode, the system switches to the following simulated input image:

- The external EMERGENCY STOP is not active.
- The safety gate is open.
- No safety stop 1 has been requested.
- No safety stop 2 has been requested.
- No safe operational stop has been requested.
- Only for VKR C4: E2/E22 is closed.

If SafeOperation or SafeRangeMonitoring is used, Start-up mode also influences other signals.

Information about the effects of Start-up mode in conjunction with SafeOperation or SafeRangeMonitoring can be found in the documentation **SafeOperation** and **SafeRangeMonitoring**.

Precondition

- T1 mode
- In the case of VKR C4: no E2/E22/E7 signals are activated via a USB stick or retrofit interface.
- In the case of RoboTeam: the local smartPAD is used.
- If the Ethernet safety interface is used: No connection to a higher-level safety controller

Procedure

- In the main menu, select **Start-up** > **Service** > **Start-up mode**.

Menu	Description
	Start-up mode is active. Touching the menu item deactivates the mode.
	Start-up mode is not active. Touching the menu item activates the mode.

5.11 Checking the activation of the positionally accurate robot model

Description

If a positionally accurate robot is used, it must be checked that the positionally accurate robot model is activated.

In the case of positionally accurate robots, position deviations resulting from workpiece tolerances and elastic effects of the individual robots are compensated for. The positionally accurate robot positions the programmed TCP anywhere in the Cartesian workspace within the tolerance limits. The model parameters of the positionally accurate robot are determined at a calibration station and permanently saved on the robot (RDC).

The positionally accurate robot model is only valid for the robot as delivered.
Following conversion or retrofitting of the robot, e.g. with an arm extension or a new wrist, the robot must be recalibrated.

Functions

A positionally accurate robot has the following functions:

- Increased positioning accuracy, approximately by the factor 10
- Increased path accuracy

A precondition for the increased positioning and path accuracy is the correct input of the load data into the robot controller.

- Simplified transfer of programs if the robot is exchanged (no reteaching)
- Simplified transfer of programs after offline programming with WorkVisual (no reteaching)

Procedure

1. In the main menu, select **Help** > **Info**.
2. Check on the **Robot** tab that the positionally accurate robot model is activated. (= specification **Positionally accurate robot**).

5.12 Activating palletizing mode

Description

Only relevant for palletizing robots with 6 axes!

In the case of palletizing robots with 6 axes, palletizing mode is deactivated by default and must be activated. When palletizing mode is active, A4 is locked at 0° and the mounting flange is parallel to the floor.

Precondition

- The robot is mastered.
- There is no load on the robot; i.e. there is no tool, workpiece or supplementary load mounted.

Procedure

- Activate palletizing mode in the program as follows:

```
$PAL_MODE = TRUE
```

Alternative procedure

1. Set $PAL_MODE to TRUE via the variable correction function.
2. The following message is displayed: *Palletizing mode: Move axis A4 [direction] into position*.

   Move A4 in the direction specified in the message (plus or minus).
3. Once A4 has reached its position (0°), the following message is displayed: *Palletizing mode: Move axis A5 [direction] into position*.

   Move A5 in the direction specified in the message (plus or minus).

   Once A5 has reached its position (90°), the message disappears.

The labels **A4** and **A5** next to the jog keys now disappear. These axes can no longer be jogged.

Restrictions

- After every cold restart of the robot controller, $PAL_MODE is automatically set to FALSE.

  Recommendation: Integrate $PAL_MODE = TRUE into the initialization section of all programs for the palletizing robot.

- In the case of robots with palletizing mode active, payload determination with KUKA.LoadDataDetermination is not possible.

  WARNING

  In the case of robots with palletizing mode active, payload determination with KUKA.LoadDataDetermination must not be carried out. Injuries or damage to property may result.

- If palletizing mode is active, the robot cannot be mastered. If mastering is nonetheless required, proceed as follows:
  1. Remove all loads from the robot.
  2. Set $PAL_MODE to FALSE via the variable correction function.
  3. Master the robot.
  4. Set $PAL_MODE to TRUE.

     (Not necessary if $PAL_MODE = TRUE is in the initialization section of all programs for the palletizing robot.)
  5. Move the robot to the palletizing position.
  6. Re-attach all loads to the robot.

5.13 Mastering

Overview

Every robot must be mastered. Only if the robot has been mastered can it move to programmed positions and be moved using Cartesian coordinates. During mastering, the mechanical position and the electronic position of the robot are aligned. For this purpose, the robot is moved to a defined mechanical position, the mastering position. The encoder value for each axis is then saved.

The mastering position is similar, but not identical, for all robots. The exact positions may even vary between individual robots of a single robot type.

Fig. 5-6: Mastering position – approximate position

A robot must be mastered in the following cases:

Case	Comment
During commissioning	- - -
After maintenance work during which the robot loses its mastering, e.g. exchange of motor or RDC	(>>> *5.13.8 "Reference mastering" Page 159*)
When the robot has been moved without the robot controller (e.g. with the release device)	- - -
After exchanging a gear unit	Before carrying out a new mastering procedure, the old mastering data must first be deleted! Mastering data are deleted by manually unmastering the axes. (>>> *5.13.10 "Manually unmastering axes" Page 166*)
After an impact with an end stop at more than 250 mm/s	
After a collision.	

5.13.1 Mastering methods

Overview

The mastering methods that can be used for a robot depend on the type of gauge cartridge with which it is equipped. The types differ in terms of the size of their protective caps.

Type of gauge cartridge	Mastering methods
Gauge cartridge for **SEMD** (Standard Electronic Mastering Device) Protective cap with fine thread, M20	Mastering with the probe, type **SEMD** (>>> *5.13.5 "Mastering with the SEMD" Page 151*)
	Mastering with the dial gauge (>>> *5.13.6 "Mastering with the dial gauge" Page 157*)
	Reference mastering Only for mastering after certain maintenance work
Gauge cartridge for **MEMD** (Micro Electronic Mastering Device) Protective cap with fine thread, M8	Mastering with the probe, type **MEMD** On A6 in certain cases: mastering to the mark (>>> *5.13.9 "Mastering with the MEMD and mark" Page 160*)

SEMD/MEMD

SEMD and/or MEMD are contained in the KUKA mastering kit. There are several variants of the mastering kit.

Fig. 5-7: Mastering kit with SEMD and MEMD

1 Mastering box
2 Screwdriver for MEMD
3 MEMD
4 SEMD
5 Cables

The thinner cable is the signal cable. It connects the SEMD or MEMD to the mastering box.

The thicker cable is the EtherCAT cable. It is connected to the mastering box and to the robot at X32.

NOTICE

- Leave the signal cable connected to the mastering box and disconnect it as little as possible. The pluggability of the M8 sensor connector is limited. Frequent connection/disconnection can result in damage to the connector.
- In the case of probes to which the signal cable is not permanently attached, always screw the device onto the gauge cartridge without the signal cable. Only then may the cable be attached to the device. Otherwise, the cable could be damaged. Similarly, when removing the device, the signal cable must always be removed from the device first. Only then may the device be removed from the gauge cartridge.
- After mastering, remove the EtherCAT cable from connection X32. Failure to do so may result in interference signals or damage.

5.13.2 Moving axes to the pre-mastering position using mastering marks

Description

The axes must be moved to the pre-mastering position before every mastering operation. To do so, each axis is moved so that the mastering marks line up.

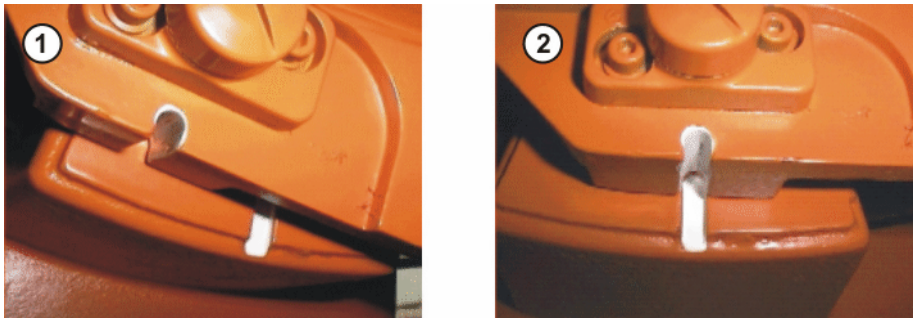


Fig. 5-8: Moving an axis to the pre-mastering position

In some cases it is not possible to align the axes using the mastering marks, e.g. because the marks can no longer be recognized due to fouling. The axes can also be mastered using the probe instead of the mastering marks.
(>>> *5.13.3 "Moving axes to the pre-mastering position using the probe" Page 148*)

The following figure shows where on the robot the mastering marks are situated. Depending on the specific robot model, the positions may deviate slightly from those illustrated.

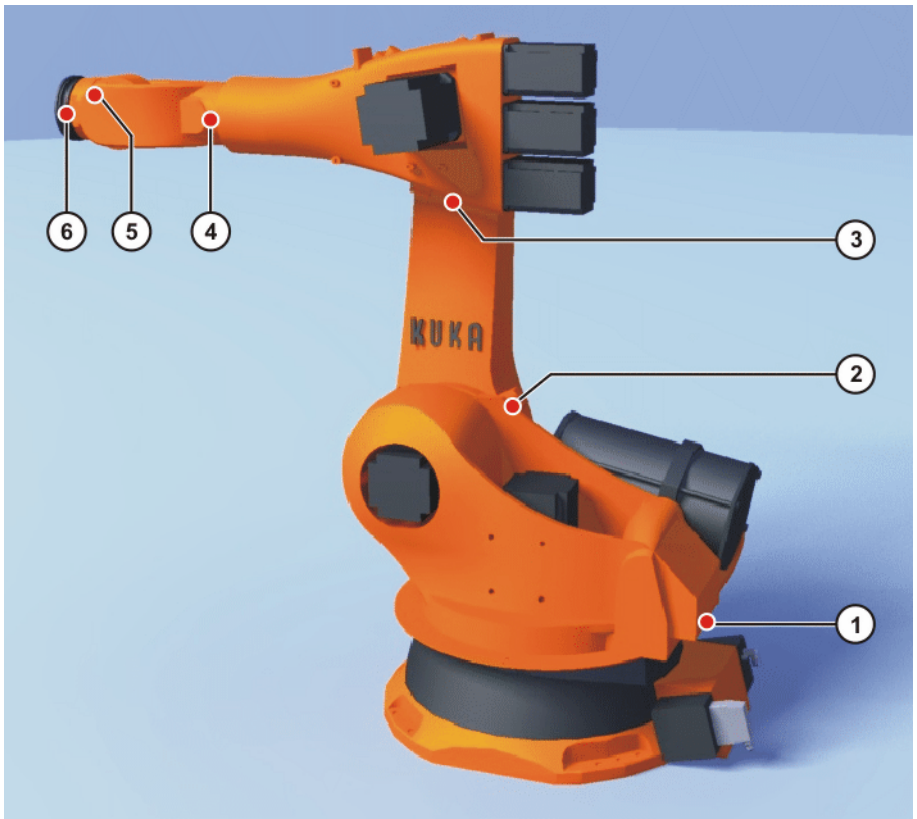


Fig. 5-9: Mastering marks on the robot

Precondition

- User rights: Function group **Jogging with the jog keys**
- T1 mode

NOTICE
Before A4 and A6 are moved to the pre-mastering position, ensure that the energy supply system – if present – is in its correct position and not rotated through 360°.

Procedure

1. Select **Axes** as the coordinate system for the jog keys.
2. Press and hold down the enabling switch.
   Axes A1 to A6 are displayed next to the jog keys.
3. Press the plus or minus jog key to move an axis in the positive or negative direction.
4. Move each axis, starting from A1 and working upwards, so that the mastering marks line up. (An exception is made for A6 of robots for which this axis is mastered using the mark.)

5.13.3 Moving axes to the pre-mastering position using the probe

Description

The axes must be moved to the pre-mastering position before every mastering operation. This is generally done using the mastering marks.

It is sometimes not possible, however, e.g. because the marks can no longer be recognized due to fouling. The axes can also be mastered using

the probe instead of the mastering marks. An LED on the smartHMI indicates when the pre-mastering position has been reached.

Precondition

- User rights of the following function groups:
  - **Mastering**
  - **Jogging with the jog keys**
- T1 mode
- No program is selected.
- The user knows the approximate pre-mastering position of the axes.

NOTICE

Before A4 and A6 are moved to the pre-mastering position, ensure that the energy supply system – if present – is in its correct position and not rotated through 360°.

Procedure

1. Jog the robot to a position in which the axes are close to their pre-mastering position. It should subsequently be possible to move them in the minus direction to the pre-mastering position.
2. In the main menu, select **Start-up** > **Master** > **EMD** > **With load correction**.

   Depending on the method for which the axes are to be aligned, the option **First mastering** or **Teach offset** or **With offset** is now selected.
3. Proceed in accordance with the instructions for the relevant mastering procedure until the probe is attached to A1 and connected via the mastering box to X32.

   Thereafter, do NOT continue to follow the description of the mastering procedure!
   i.e. do NOT press **Master** or **Learn** or **Check**!

4. The LED **EMD in mastering range** is displayed on the smartHMI. It must now be red. Observe this LED closely.

   (>>> *5.13.4 "Mastering LEDs" Page 150*)
5. Jog the robot in the minus direction. As soon as the LED switches from red to green, stop the robot.

   A1 is now in the pre-mastering position.

   The axes indicated next to the LEDs do not disappear one after the other in the usual way. This does not occur until the actual mastering.

   Do not yet master the axis. The actual mastering operation must not be carried out until all axes are in the pre-mastering position. If this is not observed, correct mastering cannot be achieved.

6. Remove the probe from the gauge cartridge as described in the mastering procedure and replace the protective cap.
7. Move the remaining axes to the pre-mastering position in the same way in ascending order. (An exception is made for A6 of robots for which this axis is mastered using the mark.)
8. Close the window containing the mastering LEDs.

9. Disconnect the EtherCAT cable from X32 and the mastering box.

NOTICE
Leave the signal cable connected to the mastering box and disconnect it as little as possible. The pluggability of the M8 sensor connector is limited. Frequent connection/disconnection can result in damage to the connector.

5.13.4 Mastering LEDs

For most mastering operations, the smartHMI displays a list of axes. There are 2 LEDs to the right of the list.

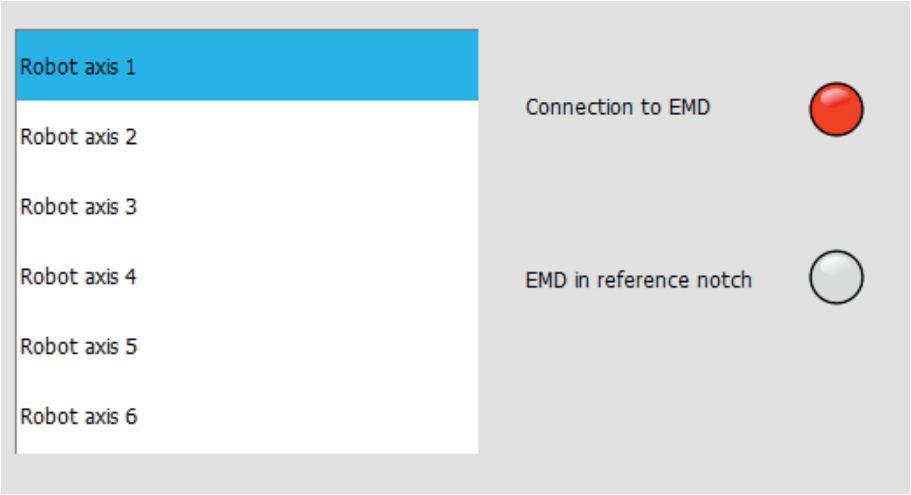


Fig. 5-10: Mastering LEDs

LED	Description
Connection to EMD	• **Red**: The probe is not connected to connection X32. • **Green**: The probe is connected to connection X32. If this LED is red, the LED **EMD in mastering range** is gray.
EMD in mastering range	• **Gray**: The probe is not connected to connection X32. • **Red**: The probe is in a position where mastering is not possible. • **Green**: The probe is either immediately next to or in the mastering notch.

The LED **EMD in mastering range** can be used to move the axes to the pre-mastering position with the aid of the probe. The pre-mastering position is reached at the moment when the LED changes from red to green during jogging in the minus direction.

(>>> *5.13.3 "Moving axes to the pre-mastering position using the probe" Page 148*)

5.13.5 Mastering with the SEMD

Overview

In SEMD mastering, the axis is automatically moved by the robot controller to the mastering position. Mastering is carried out first without and then with a load. It is possible to save mastering data for different loads.

Step	Description
1	**First mastering** (>>> *5.13.5.1 "First mastering (with SEMD)" Page 151*) First mastering is carried out without a load.
2	**Teach offset** (>>> *5.13.5.2 "Teach offset (with SEMD)" Page 154*) “Teach offset” is carried out with a load. The difference from the first mastering is saved.
3	If required: **Load mastering with offset** (>>> *5.13.5.3 "Checking load mastering with offset (with SEMD)" Page 155*) “Load mastering with offset” is carried out with a load for which an offset has already been taught. Area of application: • Checking first mastering • Restoring first mastering if it has been lost (e.g. following exchange of motor or collision). Since an offset that has been taught is retained, even if mastering is lost, the robot controller can calculate the first mastering.

5.13.5.1 First mastering (with SEMD)

Precondition

- User rights: Function group **Mastering**
- There is no load on the robot; i.e. there is no tool, workpiece or supplementary load mounted.
- All axes are in the pre-mastering position.
- No program is selected.
- T1 mode

Procedure

NOTICE

The SEMD must always be screwed onto the gauge cartridge without the signal cable attached. Only then may the cable be attached to the SEMD. Otherwise, the cable could be damaged.
Similarly, when removing the SEMD, the signal cable must always be removed from the SEMD first. Only then may the SEMD be removed from the gauge cartridge.
After mastering, remove the EtherCAT cable from connection X32. Failure to do so may result in interference signals or damage.

The SEMD actually used need not necessarily look exactly like the model illustrated in the figures. The procedure for using it is the same, however.

1. In the main menu, select **Start-up** > **Master** > **EMD** > **With load correction** > **First mastering**.

   A window opens. All axes to be mastered are displayed. The axis with the lowest number is highlighted.
2. Remove the cover from connection X32.

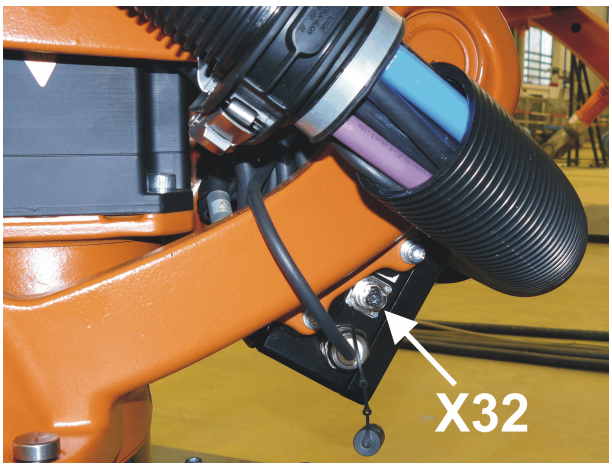


Fig. 5-11: Removing cover from X32

3. Connect the EtherCAT cable to X32 and to the mastering box.

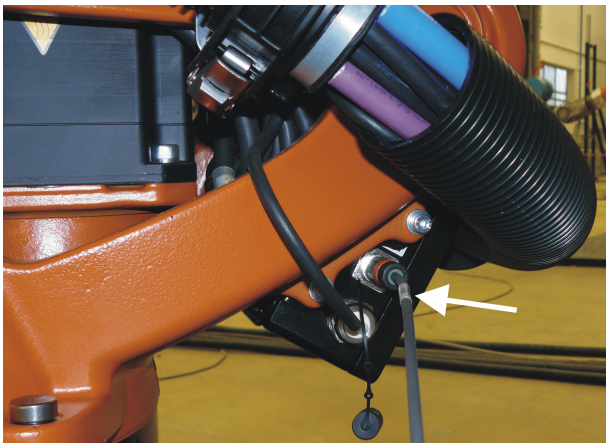


Fig. 5-12: Connecting the EtherCAT cable to X32

4. Remove the protective cap of the gauge cartridge on the axis highlighted in the window. (Turned around, the SEMD can be used as a screwdriver.)

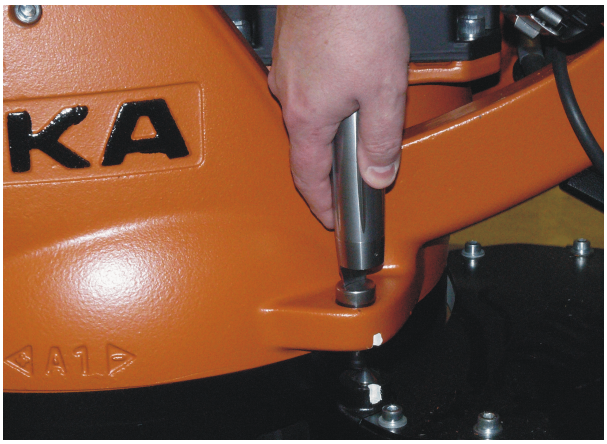


Fig. 5-13: Removing protective cap from gauge cartridge

5. Screw the SEMD onto the gauge cartridge.

Fig. 5-14: Screwing SEMD onto gauge cartridge

6. Attach the signal cable to the SEMD. It is possible to see from the cable socket which way round it has to be on the connector pins at the SEMD.

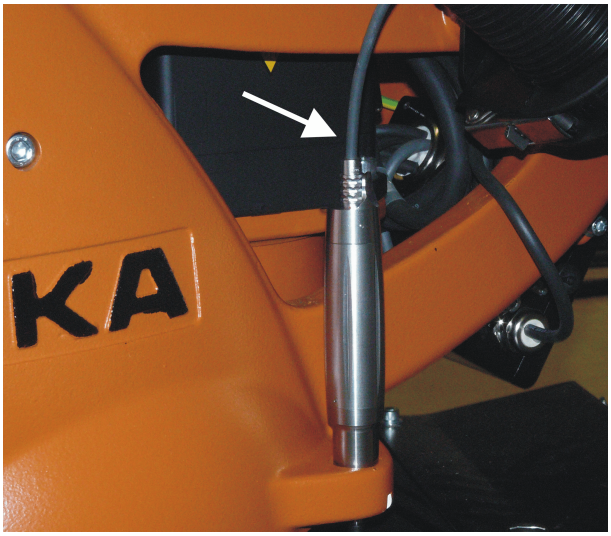


Fig. 5-15: Attaching signal cable to SEMD

7. Connect the signal cable to the mastering box if it is not already connected.
8. Press **Master**.
9. Press an enabling switch and the Start key.

   When the SEMD has passed through the reference notch, the mastering position is calculated. The robot stops automatically. The values are saved. The axis is no longer displayed in the window.
10. Remove the signal cable from the SEMD. Then remove the SEMD from the gauge cartridge and replace the protective cap.
11. Repeat steps 4 to 10 for all axes to be mastered.
12. Close the window.
13. Disconnect the EtherCAT cable from X32 and the mastering box.

NOTICE
Leave the signal cable connected to the mastering box and disconnect it as little as possible. The pluggability of the M8 sensor connector is limited. Frequent connection/disconnection can result in damage to the connector.

5.13.5.2 Teach offset (with SEMD)

Description

Teach offset is carried out with a load. The difference from the first mastering is saved.

If the robot is operated with different loads, **Teach offset** must be carried out for every load. In the case of grippers used for picking up heavy workpieces, **Teach offset** must be carried out for the gripper both with and without the workpiece.

Precondition

- User rights: Function group **Mastering**
- Same ambient conditions (temperature, etc.) as for first mastering.
- The load is mounted on the robot.
- All axes are in the pre-mastering position.
- No program is selected.
- T1 mode

Procedure

NOTICE
The SEMD must always be screwed onto the gauge cartridge without the signal cable attached. Only then may the cable be attached to the SEMD. Otherwise, the cable could be damaged. Similarly, when removing the SEMD, the signal cable must always be removed from the SEMD first. Only then may the SEMD be removed from the gauge cartridge. After mastering, remove the EtherCAT cable from connection X32. Failure to do so may result in interference signals or damage.

1. In the main menu, select **Start-up** > **Master** > **EMD** > **With load correction** > **Teach offset**.
2. Enter tool number. Confirm with **Tool OK**.

A window opens. All axes for which the tool has not yet been taught are displayed. The axis with the lowest number is highlighted.

3. Remove the cover from connection X32. Connect the EtherCAT cable to X32 and to the mastering box.
4. Remove the protective cap of the gauge cartridge on the axis highlighted in the window. (Turned around, the SEMD can be used as a screwdriver.)
5. Screw the SEMD onto the gauge cartridge.
6. Attach the signal cable to the SEMD. It is possible to see from the cable socket which way round it has to be on the connector pins at the SEMD.
7. Connect the signal cable to the mastering box if it is not already connected.
8. Press **Learn**.
9. Press an enabling switch and the Start key.

   When the SEMD has passed through the reference notch, the mastering position is calculated. The robot stops automatically. A window opens. The deviation of this axis from the first mastering is indicated in degrees and increments.
10. Confirm with **OK**. The axis is no longer displayed in the window.
11. Remove the signal cable from the SEMD. Then remove the SEMD from the gauge cartridge and replace the protective cap.
12. Repeat steps 4 to 11 for all axes to be mastered.
13. Close the window.
14. Disconnect the EtherCAT cable from X32 and the mastering box.

NOTICE
Leave the signal cable connected to the mastering box and disconnect it as little as possible. The pluggability of the M8 sensor connector is limited. Frequent connection/disconnection can result in damage to the connector.

5.13.5.3 Checking load mastering with offset (with SEMD)

Description

Area of application:

- Checking first mastering
- Restoring first mastering if it has been lost (e.g. following exchange of motor or collision). Since an offset that has been taught is retained, even if mastering is lost, the robot controller can calculate the first mastering.

An axis can only be checked if all axes with lower numbers have been mastered.

Precondition

- User rights: Function group **Mastering**
- Same ambient conditions (temperature, etc.) as for first mastering.
- A load for which **Teach offset** has been carried out is mounted on the robot.
- All axes are in the pre-mastering position.
- No program is selected.
- T1 mode

Procedure

NOTICE

The SEMD must always be screwed onto the gauge cartridge without the signal cable attached. Only then may the cable be attached to the SEMD. Otherwise, the cable could be damaged.
Similarly, when removing the SEMD, the signal cable must always be removed from the SEMD first. Only then may the SEMD be removed from the gauge cartridge.
After mastering, remove the EtherCAT cable from connection X32. Failure to do so may result in interference signals or damage.

1. In the main menu, select **Start-up** > **Master** > **EMD** > **With load correction** > **Load mastering** > **With offset**.
2. Enter tool number. Confirm with **Tool OK**.

   A window opens. All axes for which an offset has been taught with this tool are displayed. The axis with the lowest number is highlighted.
3. Remove the cover from connection X32. Connect the EtherCAT cable to X32 and to the mastering box.
4. Remove the protective cap of the gauge cartridge on the axis highlighted in the window. (Turned around, the SEMD can be used as a screwdriver.)
5. Screw the SEMD onto the gauge cartridge.
6. Attach the signal cable to the SEMD. It is possible to see from the cable socket which way round it has to be on the connector pins at the SEMD.
7. Connect the signal cable to the mastering box if it is not already connected.
8. Press **Check**.
9. Hold down an enabling switch and press the Start key.

   When the SEMD has passed through the reference notch, the mastering position is calculated. The robot stops automatically. The difference from “Teach offset” is displayed.
10. If required, press **Save** to save the values. The old mastering values are deleted.

    To restore a lost first mastering, always save the values.

    Axes A4, A5 and A6 are mechanically coupled. This means:
    If the values for A4 are deleted, the values for A5 and A6 are also deleted.
    If the values for A5 are deleted, the values for A6 are also deleted.
11. Remove the signal cable from the SEMD. Then remove the SEMD from the gauge cartridge and replace the protective cap.
12. Repeat steps 4 to 11 for all axes to be mastered.
13. Close the window.
14. Disconnect the EtherCAT cable from X32 and the mastering box.

NOTICE

Leave the signal cable connected to the mastering box and disconnect it as little as possible. The pluggability of the M8 sensor connector is limited. Frequent connection/disconnection can result in damage to the connector.

5.13.6 Mastering with the dial gauge

Description

In dial mastering, the axis is moved manually by the user to the mastering position. Mastering is always carried out with a load. It is not possible to save mastering data for different loads.

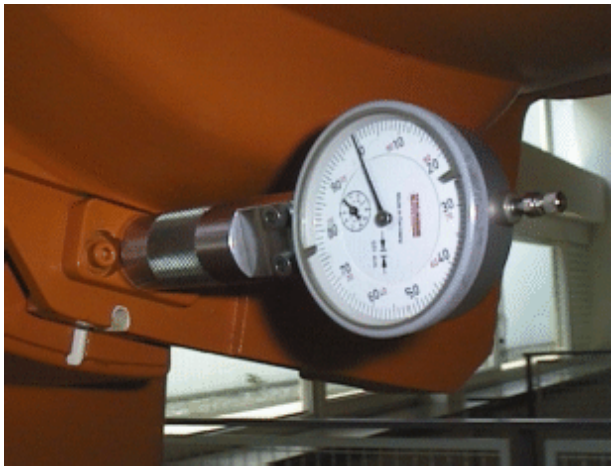


Fig. 5-16: Dial gauge

Precondition

- User rights of the following function groups:
  - **Dial mastering**
  - **General jog settings**
  - **Jogging with the jog keys**
- The load is mounted on the robot.
- All axes are in the pre-mastering position.
- **Axes** is selected as the coordinate system for jogging with the keys.
- No program is selected.
- T1 mode

Procedure

1. In the main menu, select **Start-up** > **Master** > **Dial**.

   A window opens. All axes that have not been mastered are displayed. The axis that must be mastered first is selected.
2. Remove the protective cap from the gauge cartridge on this axis and mount the dial gauge on the gauge cartridge.

   Using the Allen key, loosen the screws on the neck of the dial gauge. Turn the dial so that it can be viewed easily. Push the pin of the dial gauge in as far as the stop.

   Using the Allen key, tighten the screws on the neck of the dial gauge.
3. Reduce jog override to 1%.
4. Jog axis from “+” to “-”. At the lowest position of the reference notch, recognizable by the change in direction of the pointer, set the dial gauge to 0.

   If the axis inadvertently overshoots the lowest position, jog the axis backwards and forwards until the lowest position is reached. It is immaterial whether the axis is moved from “+” to “-” or from “-” to “+”.
5. Move the axis back to the pre-mastering position.

6. Move the axis from “+” to “-” until the pointer is about 5-10 scale divisions before zero.
7. Switch to incremental jogging.
8. Move the axis from “+” to “-” until zero is reached.

If the axis overshoots zero, repeat steps 5 to 8.

9. Press **Master**. The axis that has been mastered is removed from the window.
10. Remove the dial gauge from the gauge cartridge and replace the protective cap.
11. Switch back from incremental jogging to the normal jog mode.
12. Repeat steps 2 to 11 for all axes to be mastered.
13. Close the window.

5.13.7 Mastering external axes

Description

- KUKA external axes can be mastered using either the probe or the dial gauge.
- Non-KUKA external axes can be mastered using the dial gauge. If mastering with the probe is desired, the external axis must be fitted with gauge cartridges.

Procedure

- The procedure for mastering external axes is the same as that for mastering robot axes. Alongside the robot axes, the configured external axes now also appear in the axis selection window.

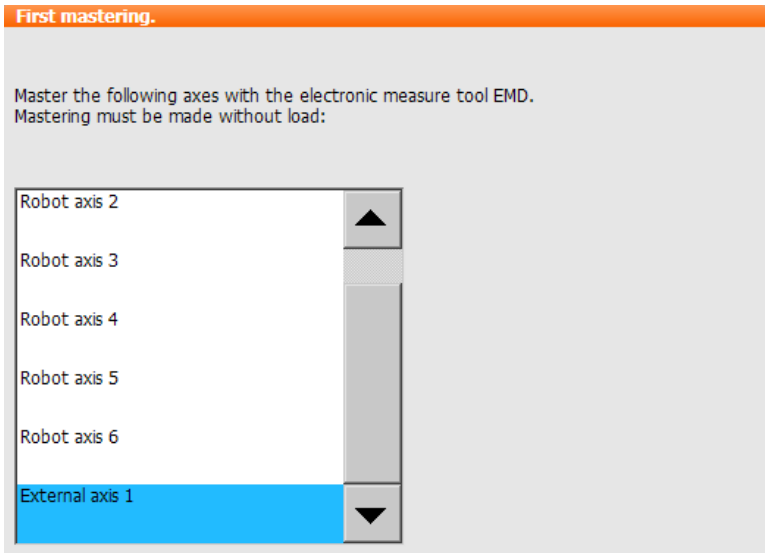


Fig. 5-17: Selection list of axes to be mastered

Mastering in the case of industrial robots with more than 2 external axes: if the system contains more than 8 axes, it may be necessary to connect the signal cable of the probe to the second RDC.

5.13.8 Reference mastering

The procedure described here must not be used when commissioning the robot.

Description

Reference mastering is suitable if maintenance work is due on a correctly mastered robot and it is to be expected that the robot will lose its mastering. Examples:

- Exchange of RDC
- Exchange of motor

The robot is moved to the $MAMES position before the maintenance work is commenced. Afterwards, the axis values of this system variable are reassigned to the robot by means of reference mastering. The state of the robot is then the same as before the loss of mastering. Taught offsets are retained. No EMD or dial gauge is required.

In the case of reference mastering, it is irrelevant whether or not there is a load mounted on the robot. Reference mastering can also be used for external axes.

Preparation

- Move the robot to the $MAMES position before commencing the maintenance work. To do so, program a point PTP $MAMES and move the robot to it. This is only possible in the user group “Expert” or higher!

WARNING

The robot must not move to the default HOME position instead of to $MAMES. $MAMES may be, but is not always, identical to the default HOME position. Only in the $MAMES position will the robot be correctly mastered by means of reference mastering. If the robot is reference mastered at any position other than $MAMES, this may result in injury and material damage.

Precondition

- User rights: Function group **Mastering**
- No program is selected.
- T1 mode
- The position of the robot was not changed during the maintenance work.
- If the RDC has been exchanged: the robot data have been transferred from the hard drive to the RDC.

  (>>> *4.19.13 "Displaying/editing robot data" Page 112*)

Procedure

1. In the main menu, select **Start-up** > **Master** > **Reference**.

   The option window **Reference mastering** is opened. All axes that have not been mastered are displayed. The axis that must be mastered first is selected.
2. Press **Master**. The selected axis is mastered and removed from the option window.
3. Repeat step 2 for all axes to be mastered.

5.13.9 Mastering with the MEMD and mark

Overview

In MEMD mastering, the axis is automatically moved by the robot controller to the mastering position. Mastering is carried out first without and then with a load. It is possible to save mastering data for different loads.

- In the case of robots with line marks on A6 instead of conventional mastering marks, A6 is mastered without MEMD.

  (>>> *5.13.9.1 "Moving A6 to the mastering position (with line mark)" Page 160*)
- In the case of robots with mastering marks on A6, A6 is mastered in the same way as the other axes.

Step	Description
1	**First mastering** (>>> *5.13.9.2 "First mastering (with MEMD)" Page 161*) First mastering is carried out without a load.
2	**Teach offset** (>>> *5.13.9.3 "Teach offset (with MEMD)" Page 164*) “Teach offset” is carried out with a load. The difference from the first mastering is saved.
3	If required: **Load mastering with offset** (>>> *5.13.9.4 "Checking load mastering with offset (with MEMD)" Page 165*) “Load mastering with offset” is carried out with a load for which an offset has already been taught. Area of application: • Checking first mastering • Restoring first mastering if it has been lost (e.g. following exchange of motor or collision). Since an offset that has been taught is retained, even if mastering is lost, the robot controller can calculate the first mastering.

5.13.9.1 Moving A6 to the mastering position (with line mark)

Description

In the case of robots with line marks on A6 instead of conventional mastering marks, A6 is mastered without MEMD.

Before mastering, A6 must be moved to its mastering position. (This means before the overall mastering process, not directly before mastering A6 itself). For this purpose, A6 has fine marks in the metal.

- To move A6 to the mastering position, the marks must be aligned exactly.

When moving to the mastering position, it is important to look at the fixed mark in a straight line from in front. If the mark is observed from the side, the movable mark cannot be aligned accurately enough. This results in incorrect mastering.

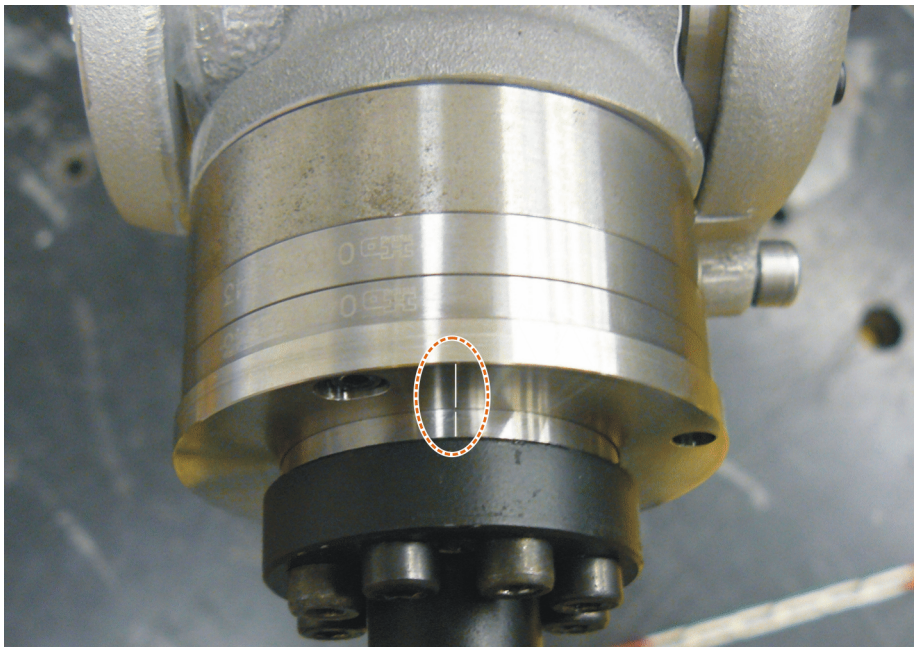


Fig. 5-18: Mastering position A6 – view from above

Mastering fixture

A mastering fixture is available for mastering A6 of the KR AGILUS. Use of this fixture is optional. Using the fixture allows mastering with greater accuracy and greater repeatability.

More information about the mastering fixture is contained in the **Mastering fixture A6** documentation.

5.13.9.2 First mastering (with MEMD)

Precondition

- User rights: Function group **Mastering**
- There is no load on the robot; i.e. there is no tool, workpiece or supplementary load mounted.
- The axes are in the pre-mastering position.

  Exception A6, if this axis has a line mark: A6 is in the mastering position.
- No program is selected.
- T1 mode

Procedure

1. In the main menu, select **Start-up** > **Master** > **EMD** > **With load correction** > **First mastering**.

   A window opens. All axes to be mastered are displayed. The axis with the lowest number is highlighted.
2. Remove the cover from connection X32.

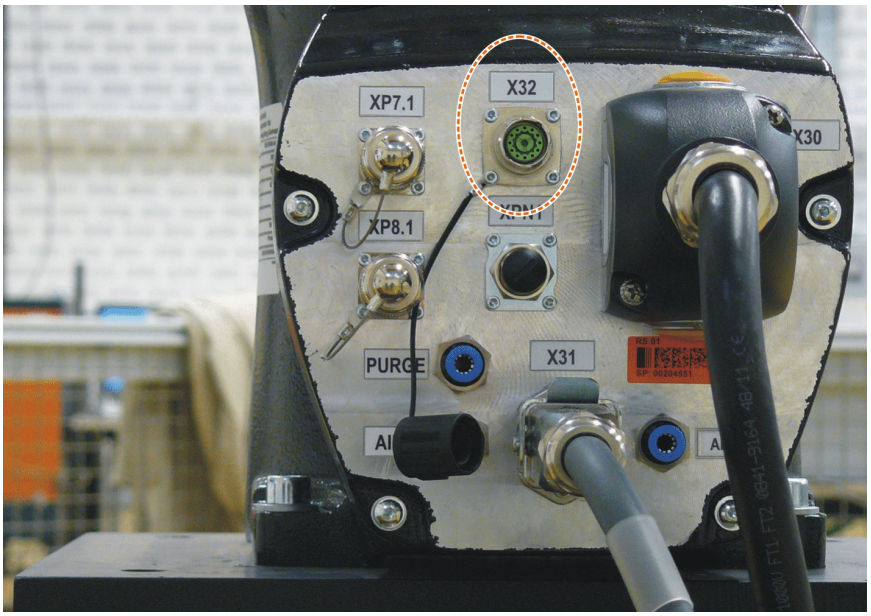


Fig. 5-19: X32 without cover

3. Connect the EtherCAT cable to X32 and to the mastering box.

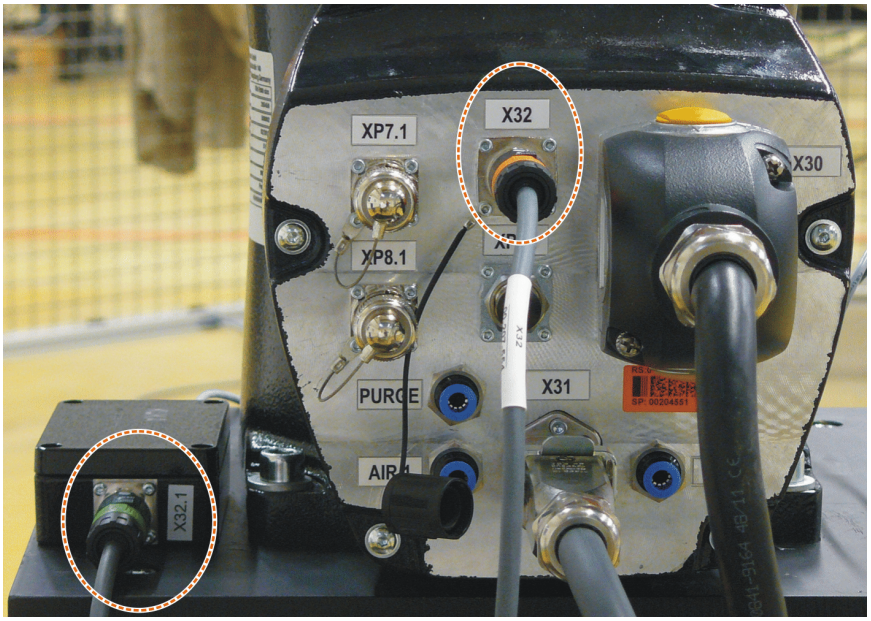


Fig. 5-20: Connecting the cable to X32

4. Remove the protective cap of the gauge cartridge on the axis highlighted in the window.

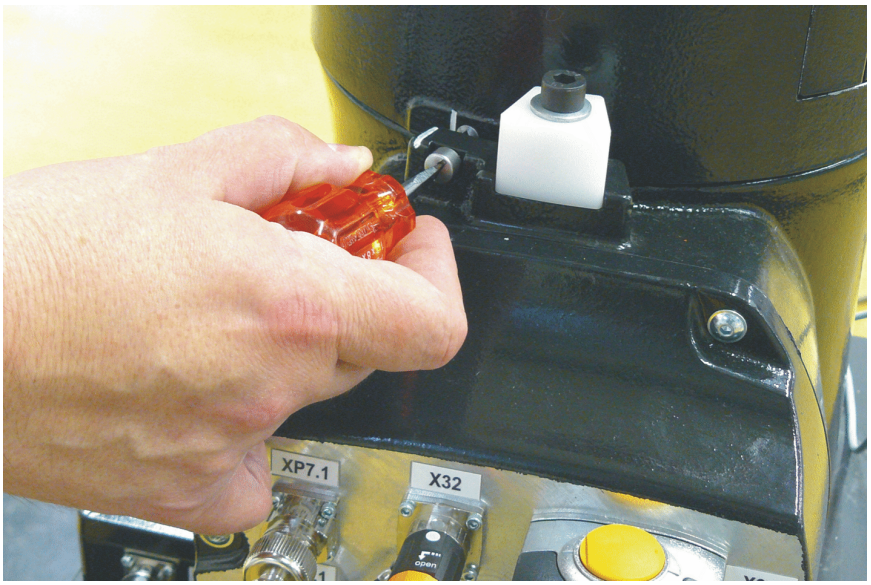


Fig. 5-21: Removing protective cap from gauge cartridge

5. Screw the MEMD onto the gauge cartridge.

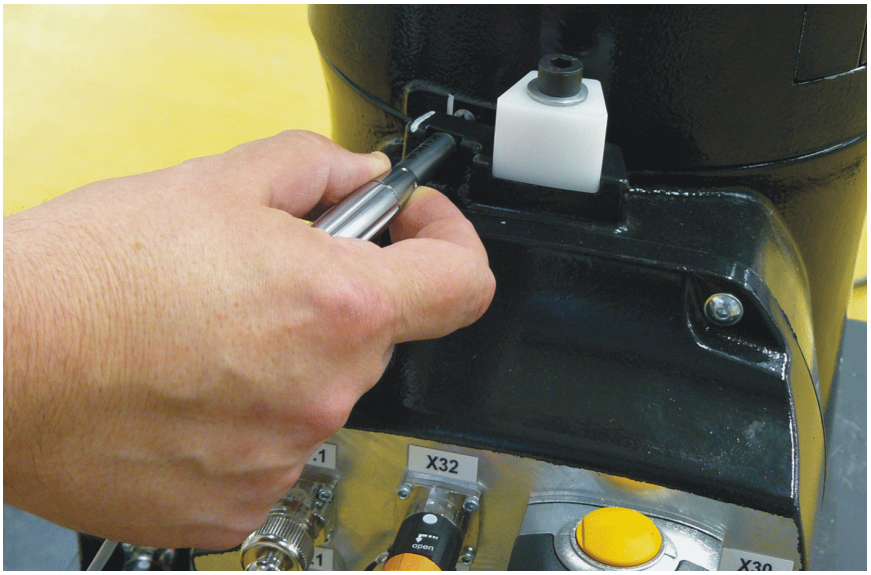


Fig. 5-22: Screwing MEMD onto gauge cartridge

6. Connect the signal cable to the mastering box if it is not already connected.
7. Press **Master**.
8. Press an enabling switch and the Start key.

   When the MEMD has passed through the reference notch, the mastering position is calculated. The robot stops automatically. The values are saved. The axis is no longer displayed in the window.
9. Remove the MEMD from the gauge cartridge and replace the protective cap.
10. Repeat steps 4 to 9 for all axes to be mastered.

    Exception: Not for A6 if this axis has a line mark.
11. Close the window.
12. This step is only to be performed if A6 has a line mark.

    a. In the main menu, select **Start-up** > **Master** > **Reference**.

The option window **Reference mastering** is opened. A6 is displayed and is selected.

b. Press **Master**. A6 is mastered and removed from the option window.

c. Close the window.

13. Disconnect the EtherCAT cable from X32 and the mastering box.

NOTICE
Leave the signal cable connected to the mastering box and disconnect it as little as possible. The pluggability of the M8 sensor connector is limited. Frequent connection/disconnection can result in damage to the connector.

5.13.9.3 Teach offset (with MEMD)

Description

Teach offset is carried out with a load. The difference from the first mastering is saved.

If the robot is operated with different loads, **Teach offset** must be carried out for every load. In the case of grippers used for picking up heavy workpieces, **Teach offset** must be carried out for the gripper both with and without the workpiece.

Precondition

- User rights: Function group **Mastering**
- Same ambient conditions (temperature, etc.) as for first mastering.
- The load is mounted on the robot.
- The axes are in the pre-mastering position.

  Exception A6, if this axis has a line mark: A6 is in the mastering position.
- No program is selected.
- T1 mode

Procedure

1. Select **Start-up** > **Master** > **EMD** > **With load correction** > **Teach offset** in the main menu.
2. Enter tool number. Confirm with **Tool OK**.

   A window opens. All axes for which the tool has not yet been taught are displayed. The axis with the lowest number is highlighted.
3. Remove the cover from connection X32.
4. Connect the EtherCAT cable to X32 and to the mastering box.
5. Remove the protective cap of the gauge cartridge on the axis highlighted in the window.
6. Screw the MEMD onto the gauge cartridge.
7. Connect the signal cable to the mastering box if it is not already connected.
8. Press **Learn**.
9. Press an enabling switch and the Start key.

   When the MEMD has passed through the reference notch, the mastering position is calculated. The robot stops automatically. A window opens. The deviation of this axis from the first mastering is indicated in degrees and increments.

10. Confirm with **OK**. The axis is no longer displayed in the window.
11. Remove the MEMD from the gauge cartridge and replace the protective cap.
12. Repeat steps 5 to 11 for all axes to be mastered.

    Exception: Not for A6 if this axis has a line mark.
13. Close the window.
14. This step is only to be performed if A6 has a line mark.
    a. In the main menu, select **Start-up** > **Master** > **Reference**.

       The option window **Reference mastering** is opened. A6 is displayed and is selected.
    b. Press **Master**. A6 is mastered and removed from the option window.
    c. Close the window.
15. Disconnect the EtherCAT cable from X32 and the mastering box.

NOTICE
Leave the signal cable connected to the mastering box and disconnect it as little as possible. The pluggability of the M8 sensor connector is limited. Frequent connection/disconnection can result in damage to the connector.

5.13.9.4 Checking load mastering with offset (with MEMD)

Description

Area of application:

- Checking first mastering
- Restoring first mastering if it has been lost (e.g. following exchange of motor or collision). Since an offset that has been taught is retained, even if mastering is lost, the robot controller can calculate the first mastering.

An axis can only be checked if all axes with lower numbers have been mastered.

In the case of robots where A6 has a line mark, the value determined for this axis is not displayed, i.e. first mastering cannot be checked for A6. It is possible to restore lost first mastering, however.

Precondition

- User rights: Function group **Mastering**
- Same ambient conditions (temperature, etc.) as for first mastering.
- A load for which **Teach offset** has been carried out is mounted on the robot.
- The axes are in the pre-mastering position.

  Exception A6, if this axis has a line mark: A6 is in the mastering position.
- No program is selected.
- T1 mode

Procedure

1. In the main menu, select **Start-up** > **Master** > **EMD** > **With load correction** > **Master load** > **With offset**.
2. Enter tool number. Confirm with **Tool OK**.

A window opens. All axes for which an offset has been taught with this tool are displayed. The axis with the lowest number is highlighted.

3. Remove the cover from connection X32.
4. Connect the EtherCAT cable to X32 and to the mastering box.
5. Remove the protective cap of the gauge cartridge on the axis highlighted in the window.
6. Screw the MEMD onto the gauge cartridge.
7. Connect the signal cable to the mastering box if it is not already connected.
8. Press **Check**.
9. Hold down an enabling switch and press the Start key.

   When the MEMD has passed through the reference notch, the mastering position is calculated. The robot stops automatically. The difference from “Teach offset” is displayed.
10. If required, press **Save** to save the values. The old mastering values are deleted.

    To restore a lost first mastering, always save the values.

    Axes A4, A5 and A6 are mechanically coupled. This means:
    If the values for A4 are deleted, the values for A5 and A6 are also deleted.
    If the values for A5 are deleted, the values for A6 are also deleted.

11. Remove the MEMD from the gauge cartridge and replace the protective cap.
12. Repeat steps 5 to 11 for all axes to be mastered.

    Exception: Not for A6 if this axis has a line mark.
13. Close the window.
14. This step is only to be performed if A6 has a line mark.
    a. In the main menu, select **Start-up** > **Master** > **Reference**.

       The option window **Reference mastering** is opened. A6 is displayed and is selected.
    b. Press **Master** to restore lost first mastering. A6 is removed from the option window.
    c. Close the window.
15. Disconnect the EtherCAT cable from X32 and the mastering box.

NOTICE

Leave the signal cable connected to the mastering box and disconnect it as little as possible. The pluggability of the M8 sensor connector is limited. Frequent connection/disconnection can result in damage to the connector.

5.13.10 Manually unmastering axes

Description

The mastering values of the individual axes can be deleted. The axes do not move during unmastering.

Axes A4, A5 and A6 are mechanically coupled. This means:
If the values for A4 are deleted, the values for A5 and A6 are also deleted.
If the values for A5 are deleted, the values for A6 are also deleted.

NOTICE

The software limit switches of an unmastered robot are deactivated. The robot can hit the end stop buffers, thus damaging the robot and making it necessary to exchange the buffers. An unmastered robot must not be jogged, if at all avoidable. If it must be jogged, the jog override must be reduced as far as possible.

Precondition

- User rights: Function group **Mastering**
- T1 mode
- No program is selected.

Procedure

1. In the main menu, select **Start-up** > **Master** > **Unmaster**. A window opens.
2. Select the axis to be unmastered.
3. Press **Unmaster**. The mastering data of the axis are deleted.
4. Repeat steps 2 and 3 for all axes to be unmastered.
5. Close the window.

5.14 Modifying software limit switches

There are 2 ways of modifying the software limit switches:

- Enter the desired values manually.
- Or automatically adapt the limit switches to one or more programs.

  The robot controller determines the minimum and maximum axis positions occurring in the program. These values can then be set as software limit switches.

Precondition

- User rights: Function group **General configuration**
- T1, T2 or AUT mode

Procedure

Modifying software limit switches manually:

1. In the main menu, select **Start-up** > **Service** > **Software limit switch**. The **Software limit switch** window is opened.
2. Modify the limit switches as required in the columns **Negative** and **Positive**.
3. Save the changes with **Save**.

Adapting software limit switches to a program:

1. In the main menu, select **Start-up** > **Service** > **Software limit switch**. The **Software limit switch** window is opened.
2. Click on **Auto detection**. The following message is displayed: *Auto detection is running.*

3. Start the program to which the limit switches are to be adapted. Execute the program completely and then cancel it.

   The maximum and minimum position reached by each axis is displayed in the **Software limit switch** window.
4. Repeat step 3 for all programs to which the limit switches are to be adapted.

   The maximum and minimum position reached by each axis in all executed programs is displayed in the **Software limit switch** window.
5. Once all desired programs have been executed, press **End** in the **Software limit switch** window.
6. Press **Save** to save the determined values as software limit switches.
7. If required, modify the automatically determined values manually.

   Recommendation: Reduce the determined minimum values by 5°. Increase the determined maximum values by 5°.
   This margin prevents the axes from reaching the limit switches during program execution and thus triggering a stop.

8. Save the changes with **Save**.

Description

Software limit switch window:

Fig. 5-23: Before automatic determination

Item	Description
1	Current negative limit switch
2	Current position of the axis
3	Current positive limit switch

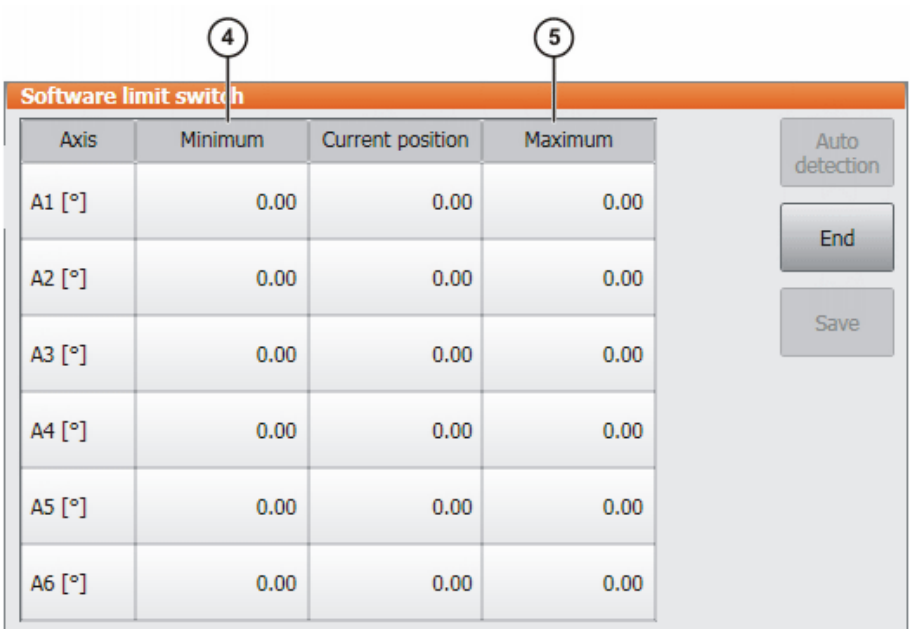


Fig. 5-24: During automatic determination

Item	Description
4	Minimum position of the axis since the start of determination
5	Maximum position of the axis since the start of determination

Buttons

The following buttons are available:

Button	Description
Auto detection	Starts the automatic determination: The robot controller writes the minimum and maximum positions adopted by the axes from now on to the columns **Minimum** and **Maximum** in the **Software limit switch** window.
End	Ends the automatic determination. Transfers the calculated minimum/maximum positions to the columns **Negative** and **Positive**, but does not yet save them.
Save	Saves the values in the columns **Negative** and **Positive** as software limit switches.

5.15 Calibration

5.15.1 Defining the tool direction

Description

By default, the X axis is defined in the system as the tool direction.

- The tool direction can be changed using the system variable $TOOL_DIRECTION.
- $TOOL_DIRECTION_LIN_CIRC defines the motions for which $TOOL_DIRECTION applies.

The tool direction applies for all tools. It is not possible to define different tool directions for different tools.

$TOOL_DIRECTION_LIN_CIRC	
FALSE (Default)	$TOOL_DIRECTION applies only for spline motions. For LIN and CIRC motions, the tool direction is the X axis and cannot be changed.
TRUE	$TOOL_DIRECTION applies for spline motions as well as for LIN and CIRC.

WARNING

The tool direction must be defined before calibration and before program creation. It cannot be modified subsequently. Failure to observe this may result in unexpected changes to the motion characteristics of the robot. Death, injuries or damage to property may result.

Precondition

- User rights: Function group **Critical KRL program changes**

Procedure

1. Open the$CUSTOM.dat file under KRC\Steu\MaDa.
2. Set $TOOL_DIRECTION_LIN_CIRC to the desired value.
   Possible values: FALSE (default); TRUE
3. Set $TOOL_DIRECTION to the desired value.
   Possible values: #X (default); #Y; #Z
4. Touch the **Close** icon. A request for confirmation is displayed, asking if the changes are to be saved.
5. Answer the query with **Yes**. The entries are saved and the window is closed.

It is not possible to modify these system variables by means of the variable correction function or by writing to the variable from the program.

5.15.2 Introduction to TOOL calibration

Description

During TOOL calibration, the user assigns a Cartesian coordinate system to a tool or workpiece that is mounted (directly or indirectly) on the mounting flange. This coordinate system is called the TOOL coordinate system.

The TOOL coordinate system has its origin at a user-defined point. This is called the TCP (Tool Center Point). The TCP is generally situated at the working point of the tool.

Advantages of TOOL calibration:

- The tool or workpiece can be moved in a straight line. This is particularly important for tools, as they can thus be moved in a straight line in the tool direction.
- The tool or workpiece can be rotated about the TCP without changing the position of the TCP.
- In program mode: the programmed velocity is maintained at the TCP along the path.

The number of TOOL coordinate systems that can be saved depends on the configuration in WorkVisual. Default: 16 TOOL coordinate systems. Variable: TOOL_DATA[1 … 16].

The following data are saved:

- X, Y, Z:
  Origin of the TOOL coordinate system relative to the FLANGE coordinate system
- A, B, C:
  Orientation of the TOOL coordinate system relative to the FLANGE coordinate system

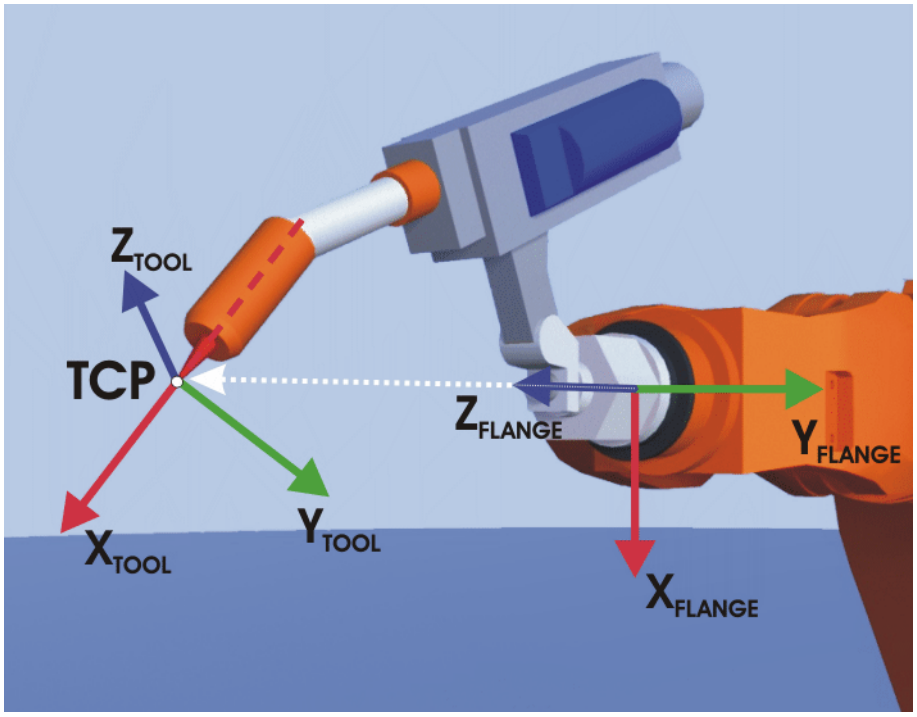


Fig. 5-25: TCP calibration principle

5.15.3 Calibrate TOOL or enter it numerically (tool/workpiece on flange)

Precondition

- User rights: Function group **Calibration**
- T1 mode
- Other preconditions dependent on the calibration method
  (>>> *5.15.7 "Overview of calibration methods" Page 178*)
  or: The values X, Y, Z, A, B and C are known.

> In the case of palletizing robots with 4 axes, the tool data must be entered numerically. The different XYZ and ABC methods cannot be used, as reorientation of these robots is highly restricted.

Procedure

1. In the main menu, select **Start-up** > **Tool/base management**.
   The **Tool/base management** window opens.
   (>>> *5.15.6 "“Tool/base management” window" Page 173*)
2. Select the **Tool Workpiece** tab.
3. Press the **Add** button. The **Edit tool** window opens.
4. Assign a number and a name.
5. In the box to the right of the name, specify what is to be calibrated:

- **Tool** or **Workpiece**

6. In the **Transformation** area, define the values for the origin (X, Y, Z) and the orientation (A, B, C) of the TOOL coordinate system:
   - Either enter the values in the boxes.
   - Or touch the **Calibrate** button, select the calibration method and perform calibration.

   During calibration, follow the instructions on the smartHMI. The measurement points can be taught in any order. Taught measurement points can be addressed again and retaught.

   - It is also possible to enter only the values for the origin (X, Y, Z) or for the orientation (A, B, C) and to determine the remaining values by means of a calibration method.
7. Save.

   Information about this calibration is now displayed in the **Last measurement** area.

5.15.4 Introduction to BASE calibration

Description

During BASE calibration, the user assigns a Cartesian coordinate system to a work surface (or the workpiece on the work surface) or a fixed tool. This coordinate system is called the BASE coordinate system. The BASE coordinate system has its origin at a user-defined point.

Advantages of BASE calibration:

- The TCP can be jogged manually along the work surface/workpiece or along the fixed tool.
- Points can be taught relative to the BASE. If it is necessary to offset the BASE, e.g. because the work surface has been offset, the points move with it and do not need to be retaught.

The number of BASE coordinate systems that can be saved depends on the configuration in WorkVisual. Default: 32 BASE coordinate systems. Variable: BASE_DATA[1 … 32].

5.15.5 Calibrate BASE or enter it numerically (base/fixed tool)

Precondition

- User rights: Function group **Calibration**
- T1 mode
- Other preconditions dependent on the calibration method

  (>>> *5.15.7 "Overview of calibration methods" Page 178*)

  or: The values X, Y, Z, A, B and C are known.

Procedure

1. In the main menu, select **Start-up** > **Tool/base management**.

   The **Tool/base management** window opens.

   (>>> *5.15.6 "“Tool/base management” window" Page 173*)
2. Select the **Base Fixed Tool** tab.
3. Press the **Add** button. The **Edit base** window opens.
4. Assign a number and a name.

5. In the box to the right of the name, specify what is to be calibrated:
   - **Base** or **Fixed tool**
6. In the **Transformation** area, define the values for the origin (X, Y, Z) and the orientation (A, B, C) of the BASE coordinate system:
   - Either enter the values in the boxes.
   - Or touch the **Calibrate** button, select the calibration method and perform calibration.

   During calibration, follow the instructions on the smartHMI.
   The measurement points can be taught in any order. Taught measurement points can be addressed again and retaught.

   - It is also possible to enter only the values for the origin (X, Y, Z) or for the orientation (A, B, C) and to determine the remaining values by means of a calibration method.
7. Save.

   Information about this calibration is now displayed in the **Last measurement** area.

5.15.6 “Tool/base management” window

5.15.6.1 Tool/base management window – “Overview” area

The “Overview” area shows all available TOOLS, BASES and external kinematic systems, each in a separate tab.

The “Overview” area is explained here using the **Tool Workpiece** tab by way of example. The other tabs have a corresponding layout.

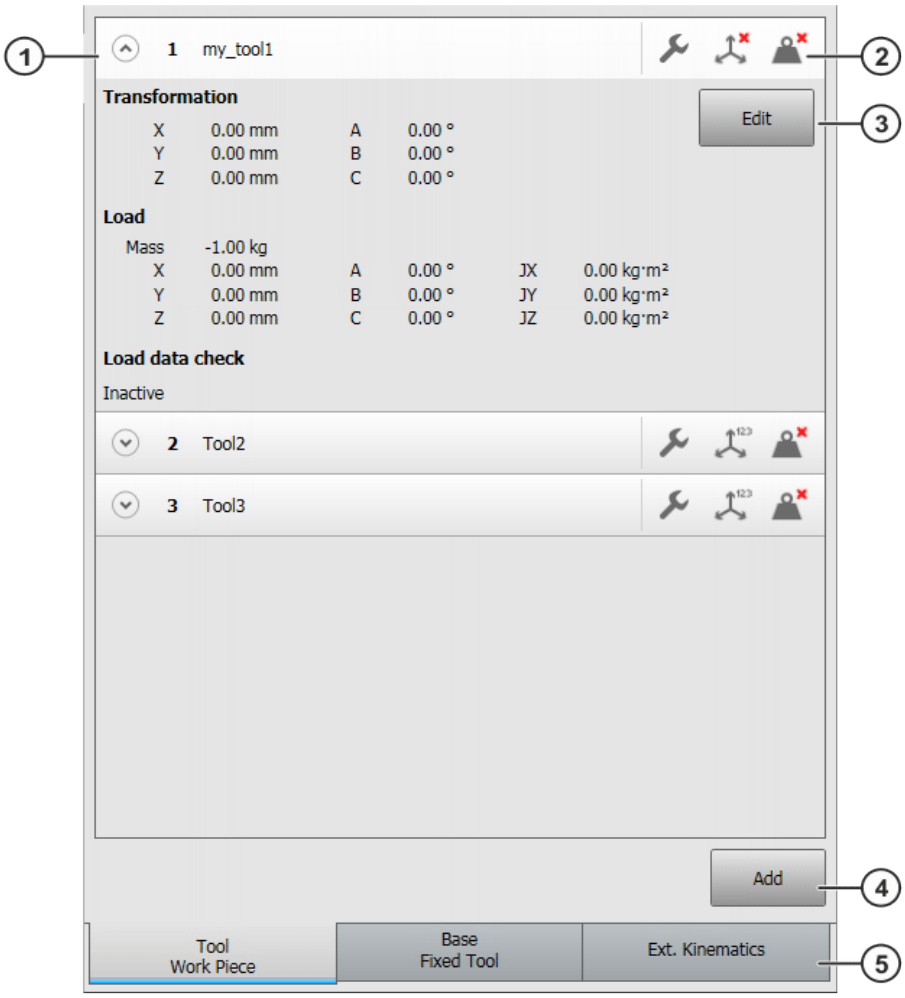


Fig. 5-26: “Overview” area

Item	Description
1	List of all created tools and workpieces. The entries can be opened by touching them.
2	The icons display information about the calibration status. Touching an icon also displays the information as text. (>>> *5.15.6.2 "Icons in the “Overview” area" Page 175*)
3	Opens the “Edit” view for this tool/workpiece. The data can be edited. (>>> *5.15.6.3 "Tool/base management window – “Editing” area" Page 176*) Required user rights: function group **Calibration**
4	Creates a new object and opens the “Edit” view. Here you can define whether the new object is to be a tool or workpiece. Required user rights: function group **Calibration**
5	Tabs for switching between TOOLS, BASES and external kinematic systems **Note: Add** is not available in **Ext. kinematic systems**. External kinematic systems cannot be added directly on the robot controller. They can only be added via WorkVisual.

5.15.6.2 Icons in the “Overview” area

Icon	Frame type
	The frame is a tool or a fixed tool.
	The frame is a base or a workpiece.
[Empty display]	The display can be empty in the following case: • The tool or base have not been created via WorkVisual or via the **Tool/base management** window, but rather in another way, e.g. via value assignment in the program. • And: the associated TOOL_TYPE[] or BASE_TYPE[] variable has not been set. **Note:** TOOL_TYPE[] or BASE_TYPE[] must still be set in this case.

Icon	Calibration
	The frame has been calibrated.
	The calibration data for the frame were entered numerically.
	The frame corresponds to $NULLFRAME.

Icon	Load data
	The load data were determined automatically.
	The load data were entered numerically.
	No load data available. The default load data are used.

Icon	Base assignment
	The base is assigned to the WORLD coordinate system.
	The base is assigned to an external kinematic system.

Icon	External kinematic system type
	Local external kinematic system
	Robot as RoboTeam participant. A number to the right of this represents the RoboTeam index.
	Conveyor

5.15.6.3 Tool/base management window – “Editing” area

The “Editing” area is explained here using the **Tool/Workpiece** tab by way of example.

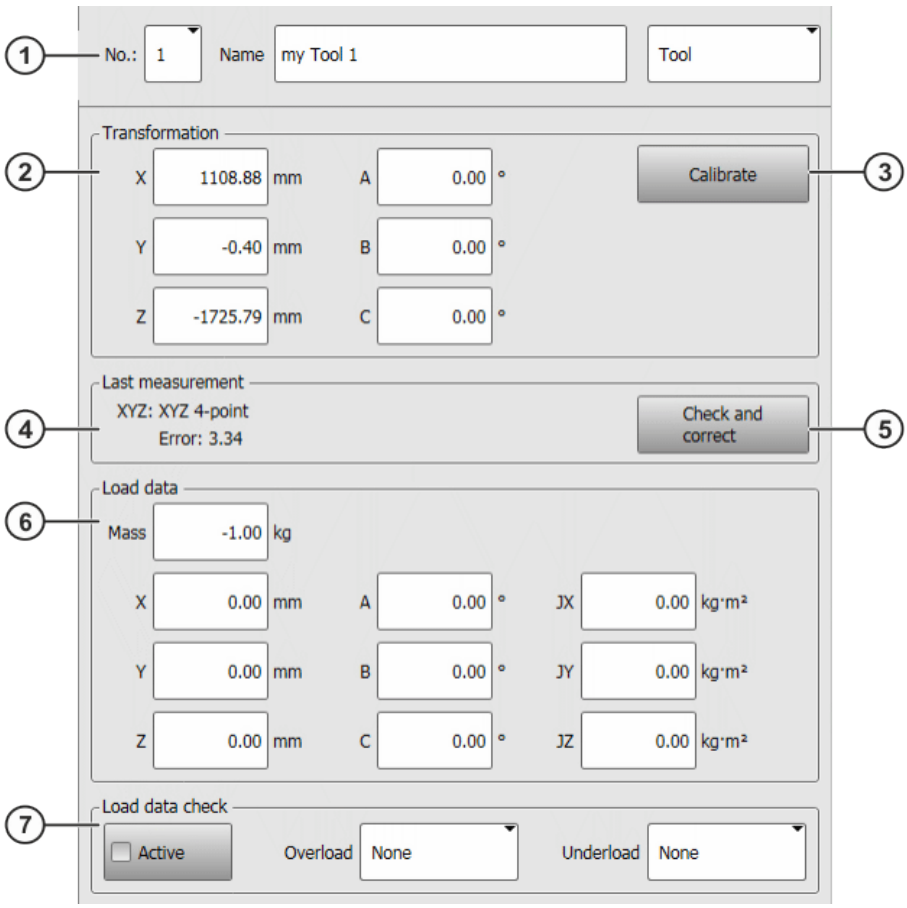


Fig. 5-27: “Editing” area

Item	Description
1	Index and name of the object The object type can be selected to the right of the name
2	Once calibration has been carried out, the transformation values are displayed here. They can also be entered numerically here.
3	Displays the possible calibration methods. Touching a method opens the “Calibration” area. A (new) calibration can be started there. (>>> *5.15.6.4 "Tool/base management window – “Calibration” area" Page 177*)
4	Information about the last calibration
5	Opens the “Calibration” area for the tool/workpiece. The existing values can be viewed in detail. If required, measurement points can be addressed again or modified and saved again if applicable.

Item	Description
6	**Tool Workpiece** tab: payload data (>>> *5.16.2 "Entering payload data numerically" Page 191*) **Base Fixed Tool** tab: assignment of BASE to the WORLD coordinate system or an external kinematic system
7	(>>> *5.16.4 "Load data verification" Page 192*)

Button	Description
Delete	Deletes the entry after a request for confirmation.
Save	Saves the changes.
Save Back	Saves the changes, closes the “Editing” area and displays the “Overview” area again.
Cancel	Closes the “Editing” area and displays the “Overview” area again.

WARNING

If an entry is deleted and this tool (or the base or the workpiece) is used in programs, the behavior of the robot during program execution changes following the deletion. Failure to take this precaution into consideration may result in death, injuries or damage to property.

5.15.6.4 Tool/base management window – “Calibration” area

The “Calibration” area is explained here using the **XYZ 4-point** method by way of example.

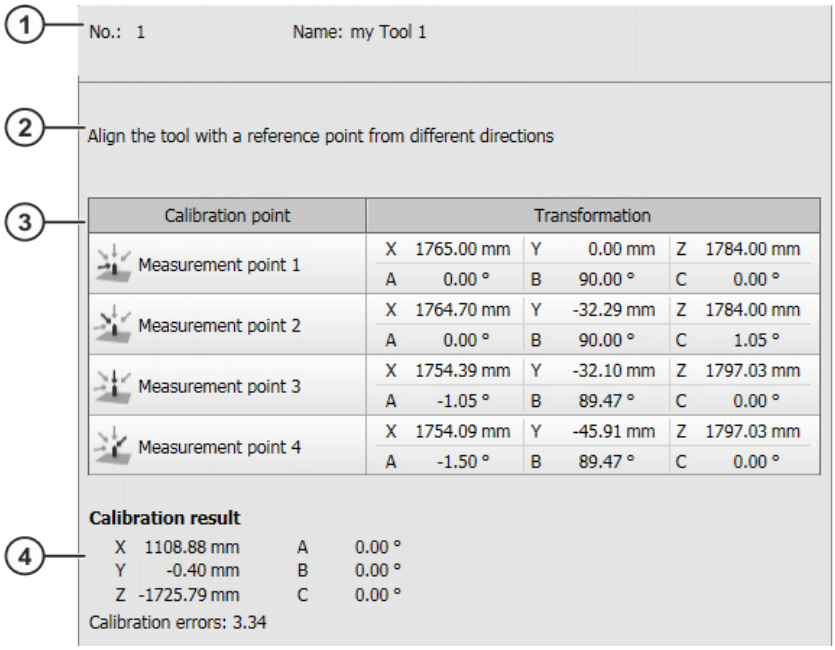


Fig. 5-28: “Calibration” view

Item	Description
1	Information about the object, e.g. index, name
2	Description of the activity to be performed Additionally, depending on the calibration method, further information or settings, e.g. for selection of a reference tool
3	List of the required measurement points If a point has already been saved, its values are displayed here.
4	Status of the current calibration As soon as a result is within the tolerances, the result is also displayed here. A potential measurement error is also only displayed here via the **XYZ 4-point** method. The display is for information purposes. The calibration can be saved if the measurement error is smaller tha the maximum permissible measurement error of 5 mm. (>>> *"Measurement result" Page 179*)

Button	Description
Touch-up	Saving a measurement point
Address (PTP)	The button is only active if a point that has already been saved is selected in the list. Touching the button causes the following message to be displayed: *Press the Start key to start the PTP motion or select "Cancel".* Once the point has been reached, this message is closed and is displayed.

5.15.7 Overview of calibration methods

The table shows which method can be used for which frame and what is calibrated using it.

	Tool on flange	Base	Fixed tool	Workpiece on flange
XYZ 4-point	X, Y, Z	- - -	- - -	- - -
XYZ Reference	X, Y, Z	- - -	- - -	- - -
XYZ		- - -	X, Y, Z	- - -
ABC world	A, B, C	- - -	A, B, C	- - -
ABC 2-point	A, B, C	- - -	A, B, C	- - -
3-point	- - -	X, Y, Z A, B, C	- - -	X, Y, Z A, B, C
Indirect	- - -	X, Y, Z A, B, C *	- - -	X, Y, Z A, B, C

* The **Indirect** method is only available if the base is assigned to an external kinematic system.

5.15.7.1 XYZ 4-point method

Use

- Calibration of X, Y and Z of a tool

The XYZ 4-point method cannot be used for palletizing robots.

Precondition

- The tool to be calibrated is mounted on the mounting flange.

Description

The TCP of the tool to be calibrated is moved to a reference point from 4 different directions. The reference point can be freely selected. The robot controller calculates the TCP from the different flange positions.

- The axis positions of the 4 measurement positions should be as different as possible. The more different they are, the more precisely the TCP can maintain its position during real reorientation.
- The 4 flange positions must not lie in a plane.

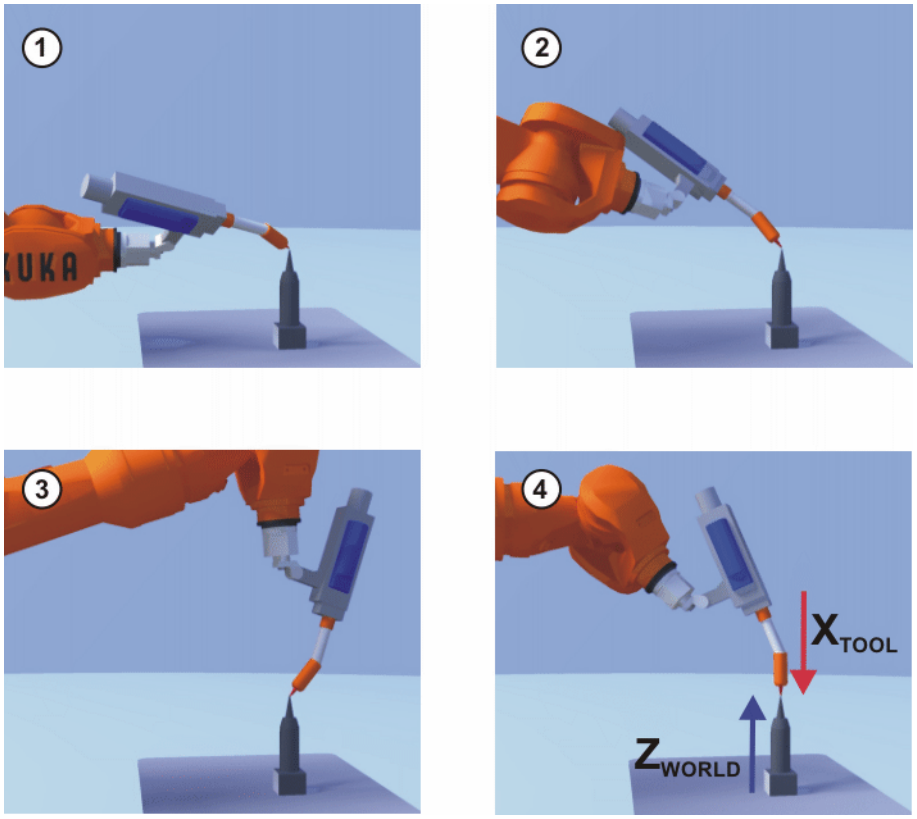


Fig. 5-29: XYZ 4-point method

Measurement result

A potential measurement error is displayed in the “Calibration” area in the **Tool/base management** window via the **XYZ 4-point** method (only by this method).

(>>> *5.15.6.4 "Tool/base management window – “Calibration” area" Page 177*)

The following factors influence the quality of a measurement result:

- Distance of TCP to flange: the shorter, the more precise

- Stiffness of the tool: the stiffer, the more precise
- Variance of the axis positions in the measurement positions: the greater the variance, the more precisely the TCP can maintain its position during real reorientation.
- Kinematic chain: inaccuracies earlier in the chain persist further down the chain.
- Mastering: incorrect mastering leads to imprecise measurement results.
- Calibrating person/“human factor”

5.15.7.2 XYZ Reference method

Use

- Calibration of X, Y and Z of a tool

Precondition

- A previously calibrated tool is mounted on the flange.
- X, Y, Z of the calibrated tool are known and available.
- The tool to be calibrated is ready to be mounted on the flange.

Description

In the case of the XYZ Reference method, a reference point is first addressed with a known tool and then with the tool to be calibrated. The robot controller compares the flange positions and calculates the TCP of the new tool.

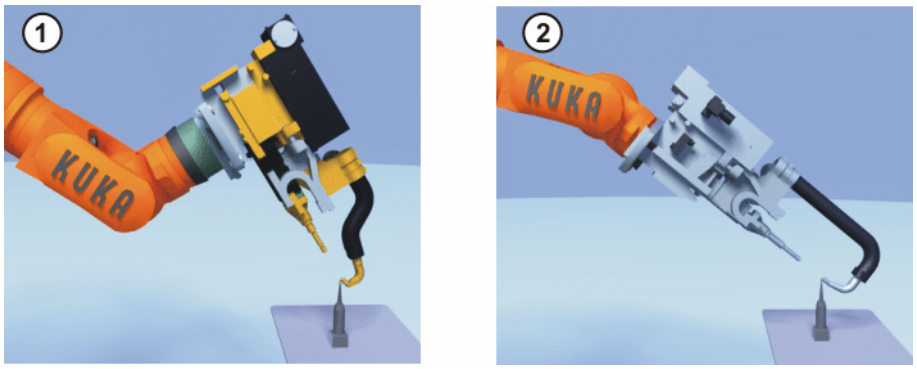


Fig. 5-30: XYZ Reference method

5.15.7.3 XYZ method

Use

- Calibration of X, Y and Z of a fixed tool

Precondition

- A previously calibrated tool is mounted on the flange.

Description

The user communicates the TCP of the fixed tool to the robot controller. This is done by moving the calibrated tool to the TCP.

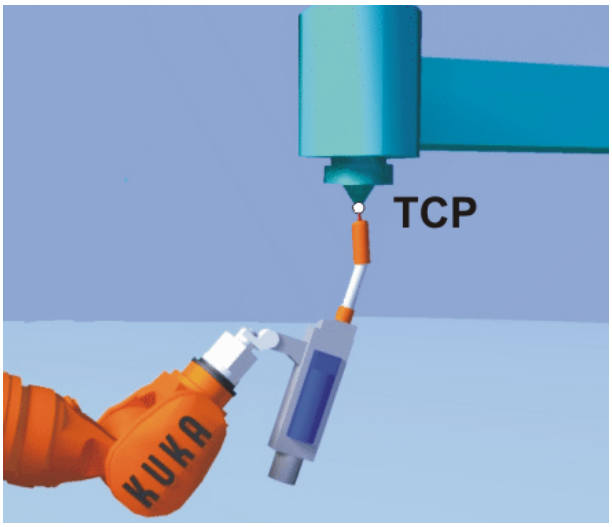


Fig. 5-31: XYZ method

5.15.7.4 ABC world method

Use

- Calibration of A, B and C of a tool
- Calibration of A, B and C of a fixed tool

Precondition

- The tool to be calibrated is mounted.

Description

Tool:

The user aligns the axes of the TOOL coordinate system parallel to the axes of the WORLD coordinate system. This communicates the orientation of the TOOL coordinate system to the robot controller.

Fixed tool:

The user aligns the FLANGE coordinate system of the calibrated tool parallel to the new coordinate system. In this way, the orientation of the coordinate system of the fixed tool is communicated to the robot controller.

Variants:

There are 2 variants of this method.

- **5D**: The user communicates the tool direction to the robot controller. By default, the tool direction is the X axis. The orientation of the other axes is defined by the system and cannot be influenced by the user.

  The system always defines the orientation of the other axes in the same way. If the tool subsequently has to be calibrated again, e.g. after a crash, it is therefore sufficient to define the tool direction again. Rotation about the tool direction need not be taken into consideration.
- **6D**: The user communicates the orientation of all 3 axes to the robot controller.

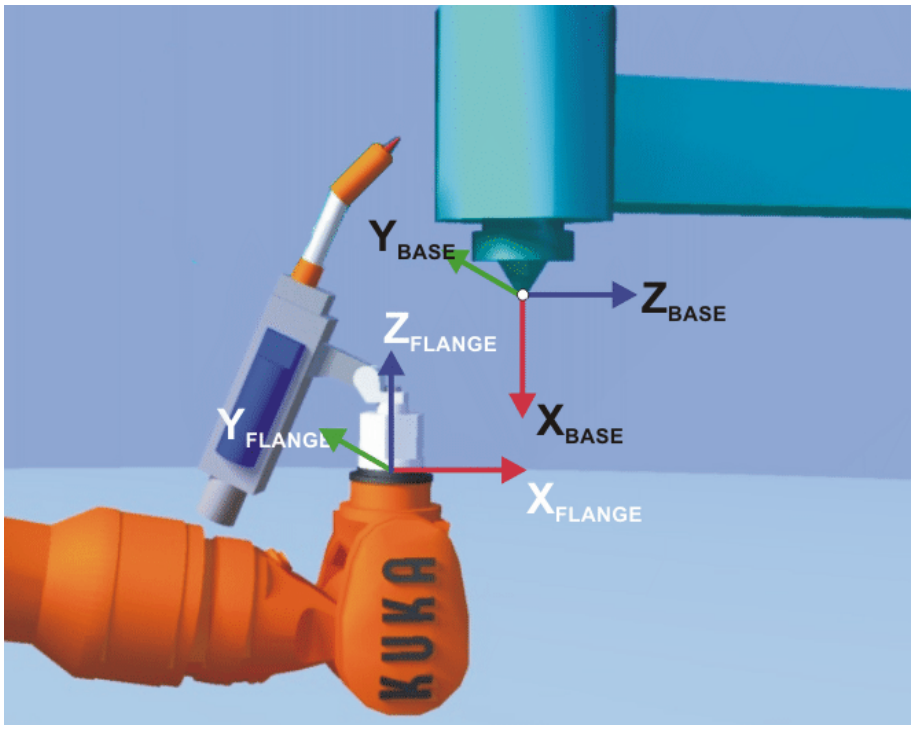


Fig. 5-32: ABC world method for a fixed tool

5.15.7.5 ABC 2-point method

Use

- Calibration of A, B and C of a tool
- Calibration of A, B and C of a fixed tool

This method is used if it is necessary to define the axis directions with particular precision.

Precondition

- The tool to be calibrated is mounted.

Description

Tool:

The axes of the TOOL coordinate system are communicated to the robot controller by moving to the following points:

1. Origin
2. Point in negative tool direction (default: X axis)
3. Point in the XY plane with a positive Y value

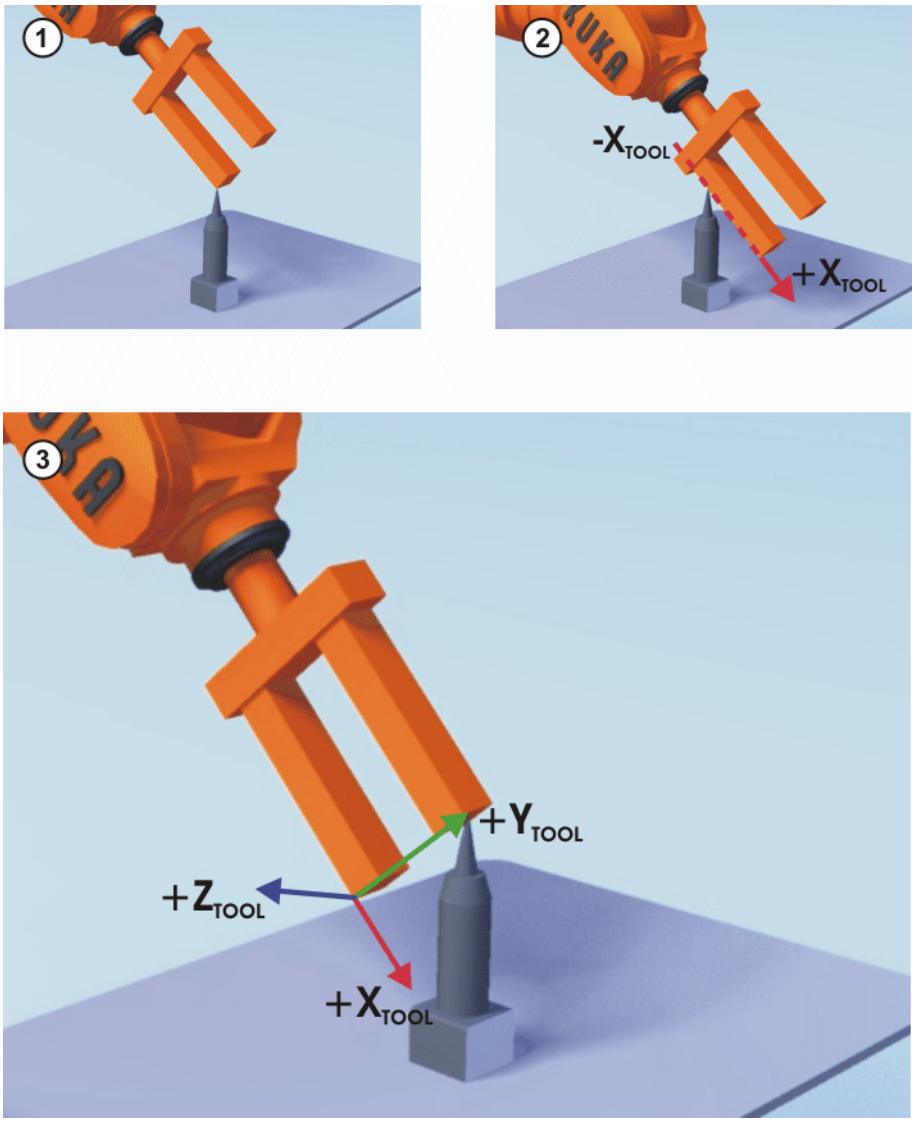


Fig. 5-33: ABC 2-point method for a tool

5.15.7.6 3-point method

Use

- Calibration of X, Y, Z, A, B, C of a base
- Calibration of X, Y, Z, A, B, C of a workpiece on the flange

Precondition

Base:

- A previously calibrated tool is mounted on the flange.

Workpiece on flange:

- The workpiece to be calibrated is mounted on the flange.
- A previously calibrated fixed tool is mounted.

Description

Base:

The robot moves to the origin and 2 further points of the new base. These 3 points unambiguously define the new base.

Workpiece on flange:

The origin and 2 further points of the workpiece are addressed. These 3 points uniquely define the workpiece.

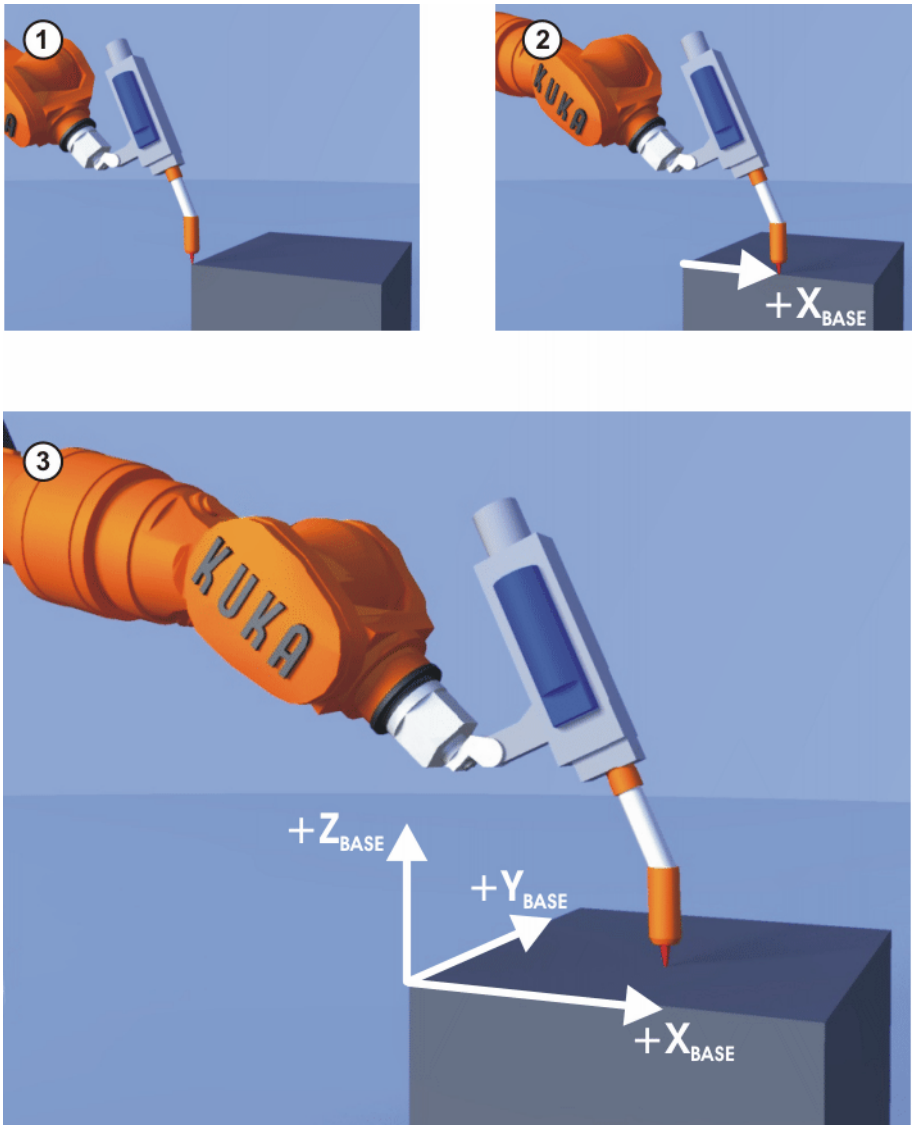


Fig. 5-34: 3-point method for a base

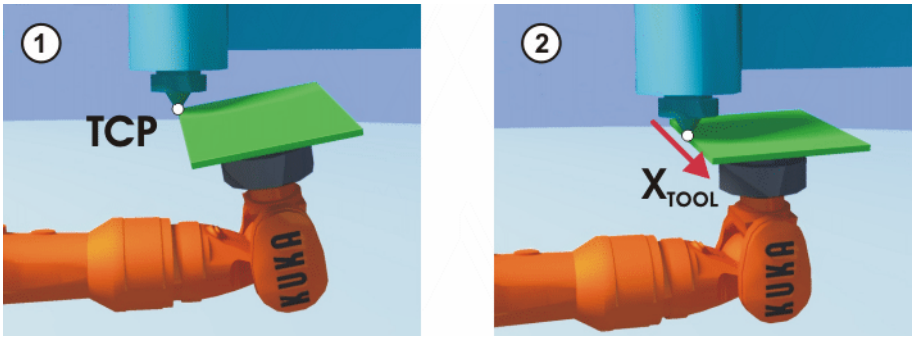


Fig. 5-35

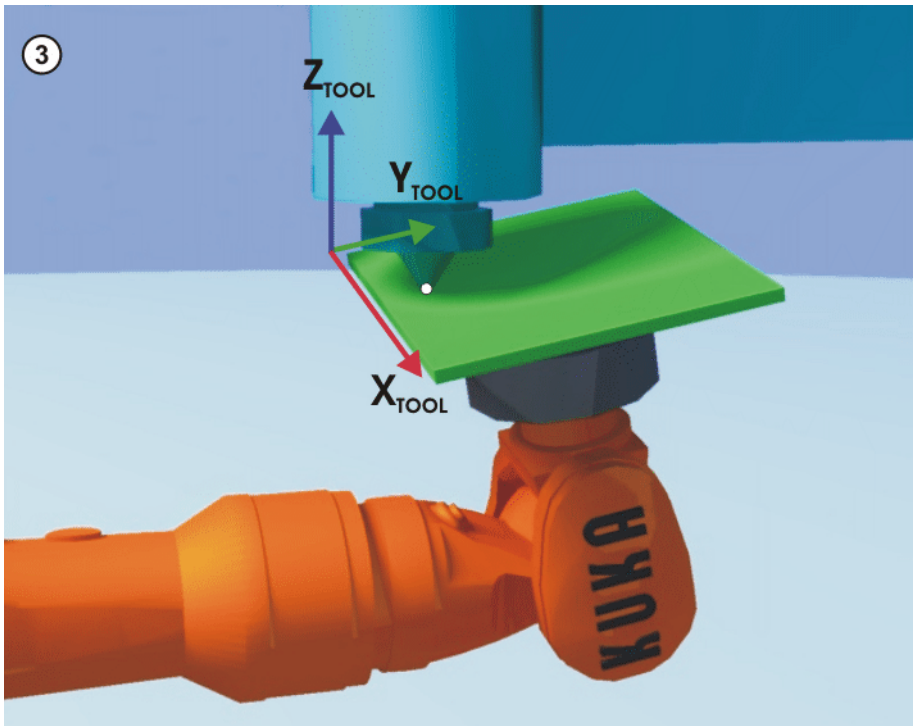


Fig. 5-36: 3-point method for a workpiece on the flange

5.15.7.7 Indirect method

Use

- Calibration of X, Y, Z, A, B, C of a base
- Calibration of X, Y, Z, A, B, C of a workpiece on the flange

The **Indirect** method is only available if the base is assigned to an external kinematic system.

Precondition

Base:

- A previously calibrated tool is mounted on the flange.
- The coordinates of 4 points in the new base are known, e.g. from CAD data. The 4 points are accessible to the TCP.

Workpiece on flange:

- The workpiece to be calibrated is mounted on the flange.
- A previously calibrated fixed tool is mounted.
- The coordinates of 4 points of the new workpiece are known, e.g. from CAD data. The locations of the 4 points are such that they can be moved to the fixed tool.

Description

Base:

The indirect method is used if it is not possible to move to the origin of the base, e.g. because it is inside a workpiece or outside the workspace of the robot.

The TCP is moved to 4 points in the base, the coordinates of which must be known. The robot controller calculates the base from these points.

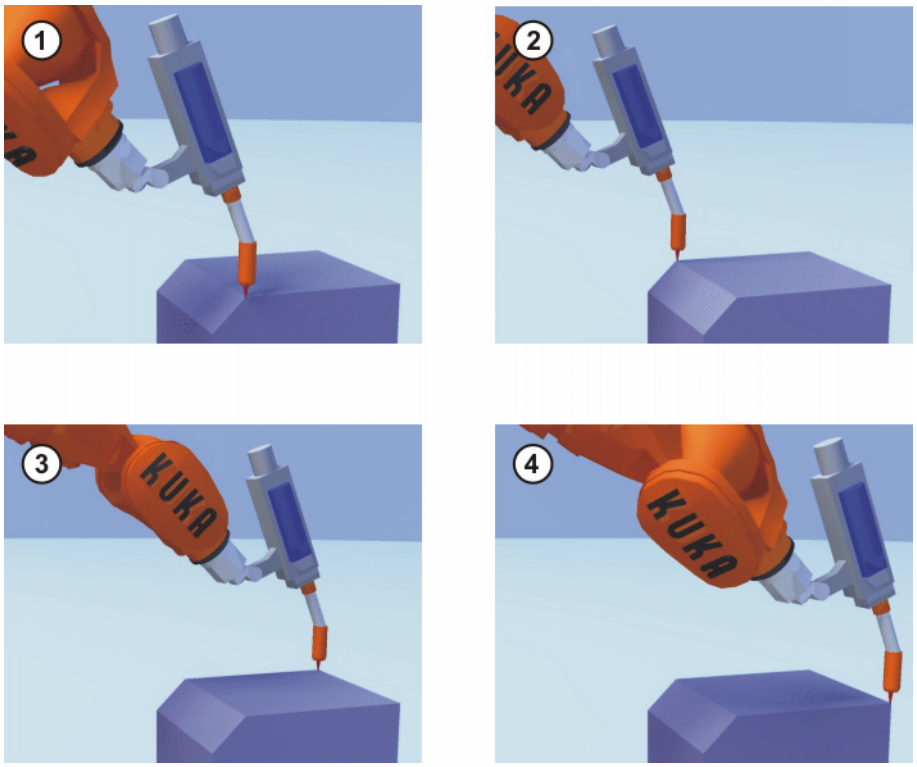


Fig. 5-37: Indirect method for a base

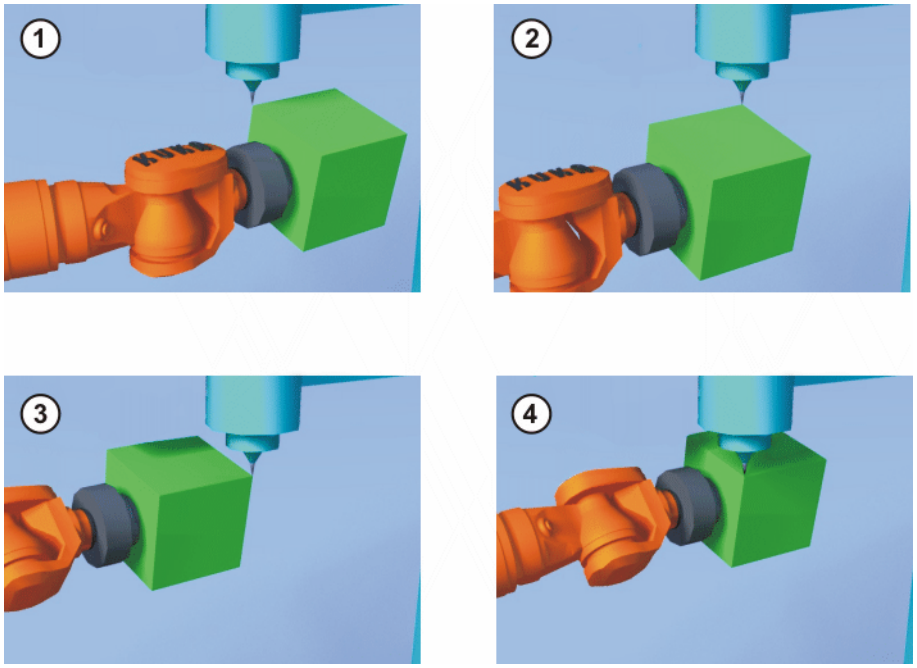


Fig. 5-38: Indirect method for a workpiece on the flange

5.15.8 Linear unit

The KUKA linear unit is a self-contained, one-axis linear unit mounted on the floor or ceiling. It is used for linear traversing of the robot and is controlled by the robot controller as an external axis.

The linear unit is a ROBROOT kinematic system. When the linear unit is moved, the position of the robot in the WORLD coordinate system changes. The current position of the robot in the WORLD coordinate system is defined by the vector $ROBROOT_C.

$ROBROOT_C consists of:

- $ERSYSROOT (static component)

  Root point of the linear unit relative to $WORLD. The root point is situated by default at the zero position of the linear unit and is not dependent on $MAMES.
- #ERSYS (dynamic component)

  Current position of the robot on the linear unit relative to the $ERSYSROOT

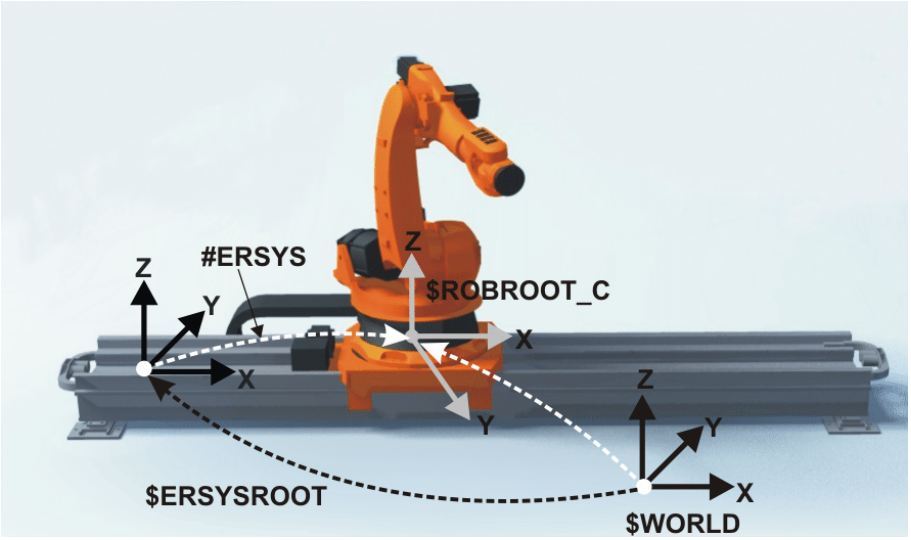


Fig. 5-39: ROBROOT kinematic system – linear unit

5.15.8.1 Checking whether the linear unit needs to be calibrated

Description

The robot is standing on the flange of the linear unit. Ideally, the ROBROOT coordinate system of the robot should be identical to the FLANGE coordinate system of the linear unit. In reality, there are often slight discrepancies which mean that positions cannot be moved to correctly. Calibration allows mathematical correction of these discrepancies. (Rotations about the direction of motion of the linear unit cannot be corrected. They do not, however, cause errors when moving to positions.)

If there are no discrepancies, the linear unit does not need to be calibrated. The following procedure can be used to determine whether calibration is required.

Precondition

- User rights: Function group **Jogging with the jog keys**
- The machine data of the linear unit have been configured and loaded into the robot controller.
- A previously calibrated tool is mounted on the mounting flange.
- No program is open or selected.
- T1 mode

Procedure

1. Align the TCP against a freely selected point and observe it.
2. Execute a Cartesian (not axis-specific) motion with the linear unit.

- If the TCP remains stationary: the linear unit does not require calibration.
- If the TCP moves: the linear unit does require calibration.

If the calibration data are already known (e.g. from CAD), they can be entered directly.

5.15.8.2 Calibrating the linear unit

Description

During calibration, the TCP of a tool that has already been calibrated is moved to a reference point 3 times.

- The reference point can be freely selected.
- The position of the robot on the linear unit from which the reference point is approached must be different all 3 times. The 3 positions must be far enough apart.

The correction values determined by the calibration are factored into the system variable $ET*x*_TFLA3.

Precondition

- User rights of the following function groups:
  - **Calibration**
  - **Jogging with the jog keys**
- The machine data of the linear unit have been configured and loaded into the robot controller.
- A previously calibrated tool is mounted on the mounting flange.
- No program is open or selected.
- T1 mode

Procedure

1. In the main menu, select **Start-up** > **Calibrate** > **Linear unit**.
2. Select the number of the tool that has already been calibrated. Confirm with **Next**.

   The robot controller detects the linear unit automatically and displays the following data:

   - **Ext. kinematic system no.**: number of the external kinematic system (1 … 6) ($EX_KIN)
   - **Axis**: number of the external axis (1 … 6) ($ET*x*_AX)
   - **Name of ext. kinematic system:** ($ET*x*_NAME)

   (If the robot controller is unable to determine these values, e.g. because the linear unit has not yet been configured, calibration cannot be continued.)
3. Move the linear unit with the jog key “+”.
4. Specify whether the linear unit has moved to “+” or “-”. Confirm with **Next**.
5. Move the TCP to the reference point.
6. Press **Calibrate**.
7. Repeat steps 5 and 6 twice, but move the linear unit first each time in order to address the reference point from different positions.

8. Press **Save**. The calibration data are saved.

NOTICE

After calibration of a linear unit, the following safety measures must be carried out:

1. Check the software limit switches of the linear unit and adapt them if required.
2. Test programs in T1.

Damage to property may otherwise result.

5.15.8.3 Entering the linear unit numerically

Precondition

- User rights of the following function groups:
  - **Calibration**
  - **Jogging with the jog keys**
- The machine data of the linear unit have been configured and loaded into the robot controller.
- No program is open or selected.
- The following numerical values are known, e.g. from CAD data:
  - Distance between the robot base flange and the origin of the ERSYSROOT coordinate system (X, Y, Z)
  - Orientation of the robot base flange relative to the ERSYSROOT coordinate system (A, B, C)
- T1 mode

Procedure

1. In the main menu, select **Start-up** > **Calibrate** > **Linear unit (numeric)**.

   The robot controller detects the linear unit automatically and displays the following data:

   - **Ext. kinematic system no.**: number of the external kinematic system (1 … 6)
   - **Axis**: number of the external axis (1 … 6)
   - **Name of ext. kinematic system:**

   (If the robot controller is unable to determine these values, e.g. because the linear unit has not yet been configured, calibration cannot be continued.)
2. Move the linear unit with the jog key “+”.
3. Specify whether the linear unit has moved to “+” or “-”. Confirm with **Next**.
4. Enter data. Confirm with **Next**.
5. Press **Save**. The calibration data are saved.

NOTICE

After calibration of a linear unit, the following safety measures must be carried out:

1. Check the software limit switches of the linear unit and adapt them if required.
2. Test programs in T1.

Damage to property may otherwise result.

5.16 Load data

Load data

Various loads can be mounted on the robot:

- Payload on the flange
- Supplementary load on axis 3
- Supplementary load on axis 2
- Supplementary load on axis 1

The load data must be communicated to the robot controller for each load. This is done by means of numerical entry. Alternatively, KUKA.LoadDataDetermination can also be used to transfer payloads to the robot controller.

The load data are factored into the calculation of the paths and accelerations and help to optimize the cycle times.

Sources

Load data can be obtained from the following sources:

- Software option KUKA.LoadDataDetermination (only for payloads)
- Manufacturer information
- Manual calculation
- CAD programs

KUKA.Load

All load data (payload and supplementary loads) must be checked with the KUKA.Load software. If the payload is determined with KUKA.LoadDataDetermination, the check with KUKA.Load follows automatically.

A sign-off sheet can be generated for the loads with KUKA.Load. KUKA.Load can be downloaded free of charge, complete with the documentation, from the KUKA website www.kuka.com.

More information is contained in the **KUKA.Load** documentation.

KUKA.LoadDataDetermination

KUKA.LoadDataDetermination can be used to calculate payloads automatically and under real conditions and to transfer them to the robot controller.

More information is contained in the **KUKA.LoadDataDetermination** documentation.

5.16.1 Load data parameters

Parameter

Parameter/unit		Description
M	kg	Mass of the load
X, **Y**, **Z**	mm	Position of the center of mass in the reference system

Parameter/unit		Description
A, **B**, **C**	Degrees	Orientation of the principal inertia axes • A: Rotation about the Z axis of the reference system Result: Coordinate system CS' • B: Rotation about the Y axis of CS' Result: CS'' • C: Rotation about the X axis of CS''
Mass moments of inertia:		
JX	kgm^2	Inertia about the X axis of the main axis system
JY	kgm^2	Inertia about the Y axis of the main axis system
JZ	kgm^2	Inertia about the Z axis of the main axis system

X, Y, Z and A, B, C unambiguously define the main axis system:

- The origin of the main axis system is the center of mass.
- A characteristic feature of the main axis system is that, among other things, the maximum possible inertia occurs about one of the 3 coordinate axes.

Reference systems for each load

Load	Reference system
Payload	FLANGE coordinate system
Supplementary load A3	FLANGE coordinate system A4 = 0°, A5 = 0°, A6 = 0°
Supplementary load A2	ROBROOT coordinate system A2 = -90°
Supplementary load A1	ROBROOT coordinate system A1 = 0°

5.16.2 Entering payload data numerically

Precondition

- User rights: Function group **Calibration**
- The payload data are known.
- The payload data have been checked with KUKA.Load and the robot is suitable for these payloads.
- The tool or workpiece for which the load data are to apply has already been created in the “Tool/base management”.

Generally, the overall unit of tool and actual workpiece has been created as a **Workpiece** in the “Tool/base management”. In this case, the payload data must also refer to the overall unit.

Procedure

1. In the main menu, select **Start-up** > **Tool/base management**.
   The **Tool/base management** window opens.

2. Open the entry for the tool or workpiece on the **Tool Workpiece** tab. Then touch the **Edit** button.
3. Enter the values under **Load data**.
   (>>> *5.16.1 "Load data parameters" Page 190*)
4. If load data verification is available (this depends on the robot type): configure as required.
   (>>> *5.16.4 "Load data verification" Page 192*)
5. Press **Save**.

5.16.3 Entering supplementary load data numerically

Precondition

- User rights: Function group **Calibration**
- The supplementary load data are known.
- The supplementary load data have been verified with KUKA.Load and are suitable for this robot type.

Procedure

1. In the main menu, select **Start-up** > **Supplementary load data**.
2. Enter the number of the axis on which the supplementary load is to be mounted. Confirm with **Next**.
3. Enter the load data.
   (>>> *5.16.1 "Load data parameters" Page 190*)
4. Confirm with **Next**.
5. Press **Save**.

5.16.4 Load data verification

Description

For many robot types, the robot controller monitors whether or not there is an overload or underload during operation.

If the verification detects an underload, for example, the robot controller reacts, e.g. by displaying a message. The reactions can be configured.

The results of the check can be polled using the system variable $LDC_RESULT. (>>> *"$LDC_RESULT" Page 194*)

Load data verification is available for those robot types for which KUKA.LoadDataDetermination can also be used. Whether or not it is available for the current robot type can be checked by means of $LDC_LOADED (TRUE = yes).

Overload	There is an overload if the actual load is significantly greater than the configured load.
Underload	There is an underload if the actual load is significantly less than the configured load.

Configuration

Load data verification is configured in the **Tool/base management** window.

Required user rights: Function group **Calibration**

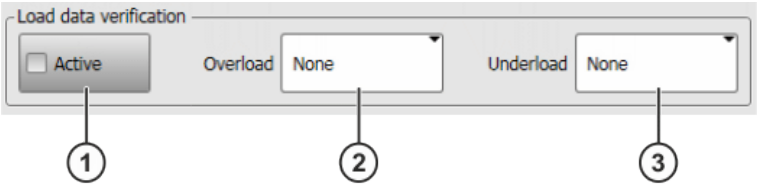


Fig. 5-40: Load data verification

Item	Description
1	**Check box active**: Verification is activated for the tool displayed in the same window. The defined reactions are carried out in the case of an overload or underload. **Check box not active**: Verification is deactivated for the tool displayed in the same window. There is no reaction in the case of an overload or underload.
2	The overload reaction can be defined here. • **None**: No reaction. • **Warning**: The robot controller generates the following status message: *Check of robot load (Tool {No.}) calculated overload.* • **Stop robot**: The robot controller generates an acknowledgement message with the same content as that generated under **Warning**. The robot stops with a STOP 2.
3	The underload reaction can be defined here. The possible reactions are analogous to those for an overload.

NULLFRAME

The load data verification cannot be configured for motions to which **NULLFRAME** has been assigned as the tool. The reactions are defined for this case and cannot be influenced by the user.

- Overload reaction: **Stop robot**

  The following acknowledgement message is generated: *Overload calculated when checking robot load (no tool defined) and the set load data.* The robot stops with a STOP 2.
- Underload reaction: **Warning**

  The following status message is generated: *Underload calculated when checking robot load (no tool defined) and the set load data.*

$LDC_CONFIG

`$LDC_CONFIG[`*Index*`]= {UNDERLOAD` *Reaction*`, OVERLOAD` *Reaction*`}`

Element	Description
Index	Type: INT Tool number
Reaction	Type: CHAR • #NONE (= **None**) • #WARNONLY (= **Warning**) • #STOPROBOT (= **Stop robot**)

Example:

```
1 ...
2 $LDC_CONFIG[1]={UNDERLOAD #NONE, OVERLOAD #NONE}
3 ...
4 $LDC_CONFIG[1]={UNDERLOAD #WARNONLY, OVERLOAD #WARNONLY}
5 ...
```

Line	Description
2	The reaction for both underload and overload is set to **None**.
4	The reaction is set to **Warning**. If there is an underload or overload, the corresponding status messages will be displayed in the message window.

$LDC_RESULT

`$LDC_RESULT[`*Index*`]=` *Result*

Element	Description
Index	Type: INT Tool number
Result	Type: CHAR • #OK: The payload is OK. (Neither overload nor underload.) • #OVERLOAD: There is an overload. • #UNDERLOAD: There is an underload. • #CHECKING • #NONE: There are currently no results, e.g. because the tool has been changed.

5.17 Exporting/importing long texts

Description

If names have been assigned to inputs/outputs, flags, etc., these names (so-called “long texts”) can be exported to a file. It is also possible to import a file with long text names. In this way, the long texts do not need to be re-entered manually for each robot after reinstallation.

The long texts can be exported to a USB stick or to the directory defined in the **Network archive path** box in the **Robot data** window. The same directories are also available as sources for the import function.

Precondition

- Either: USB stick
- Or: The target is configured in the **Network archive path** box in the **Robot data** window.

For import only:

- The long text names are present in a TXT or CSV file.
- The file is structured in such a way that it can be imported.

A file that originated as a long text export is automatically structured in such a way that it can be re-imported. If a file is to be filled with names manually, it is advisable first to assign a few dummy long texts in the robot controller, then to perform an export and fill the exported file.

Procedure

1. If a USB stick is used, connect it to the cabinet or smartPAD.
2. In the main menu, select **Start-up** > **Service** > **Long texts**. The **Long texts** window opens.
3. Select the **Export** or **Import** tab as required. Make the required settings.
4. Press the **Export** or **Import** button.

   When the import is finished, the message *Import successful.* is displayed.

   When the export is finished, the message *Export successful.* is displayed.

“Export” tab

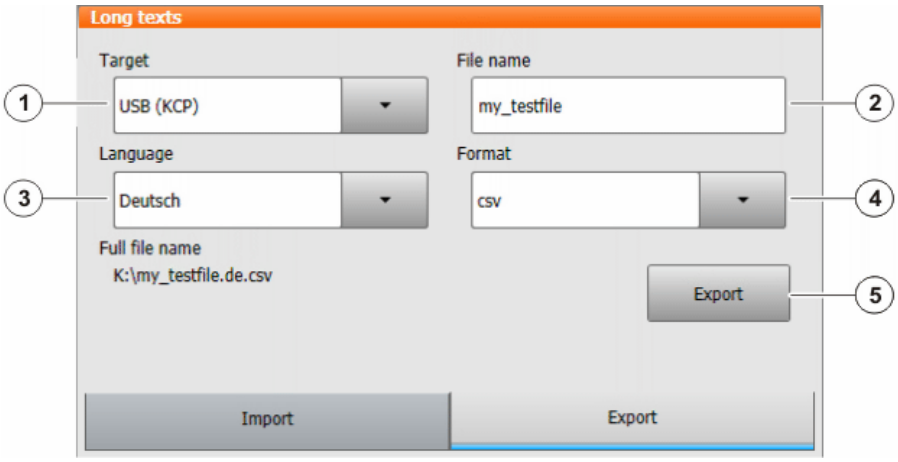


Fig. 5-41: Exporting long texts

Item	Description
1	Select the destination for the exported file. The entry **Network** is only available here if a path has been configured in the **Robot data** window.
2	Specify the desired file name. If **Network** is selected under item 1, the archive name configured in the **Robot data** window is displayed. The name can be changed here. This does not change it in the **Robot data** window. A suffix corresponding to the language selected is automatically appended to the name.
3	Select the language from which the long texts are to be exported. If, for example, the smartHMI is set to “English” and “Italiano” is selected here, a file with the suffix “it” is created. It contains the long texts that have been stored on the Italian smartHMI. It is also possible to select **All languages**.
4	Select the desired file format.
5	Starts the export.

“Import” tab

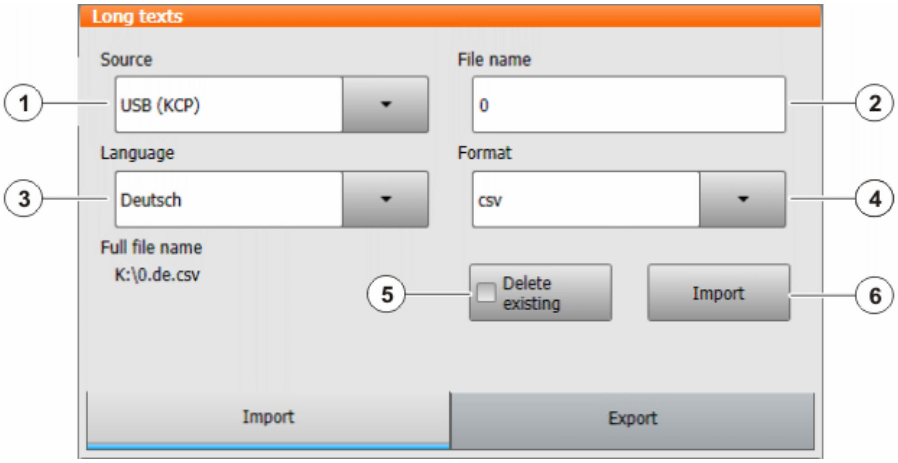


Fig. 5-42: Importing long texts

Item	Description
1	Specify the source from which files are to be imported. The entry **Network** is only available here if a path has been configured in the **Robot data** window.
2	Specify the name of the file to be imported, but without the language suffix. If **Network** is selected under item 1, the archive name configured in the **Robot data** window is displayed. The name can be changed here. This does not change it in the **Robot data** window.
3	Specify the language matching the language suffix of the file.
4	Specify the format of the file.
5	• **Activated:** All existing long texts are deleted. The contents of the file are applied. • **Deactivated:** Entries in the file overwrite existing long texts. Existing long texts for which there is no entry in the file are retained.
6	Starts the import.

5.18 Adapting the MAMES values after exchanging the in-line wrist

Description

In-line wrists of the same type can vary minimally for production reasons. When a wrist is exchanged on a robot, the relevant deviations must be communicated to the robot controller.

The values have already been determined at the factory and are provided on the identification plate of the wrist. These are the so-called MAMES values: They indicate the difference between the mechanical zero position (aligned mastering mark) and the electronic zero position (position after mastering).

Further information is also present on the identification plate, e.g. the article number. This information must also be entered.

The robot controller checks the entries for correctness by means of a checksum. The values can only be saved if they are correct.

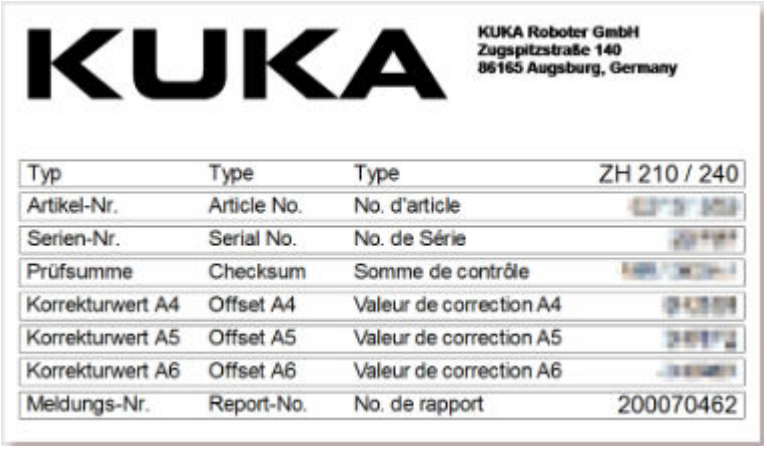
MAMES = **Ma**thematical **me**chanical **s**hift

Once the values have been saved, wrist axes A4, A5 and A6 are automatically unmastered. Mastering must be carried out.

Preparation

Retrieve the following information from the identification plate of the wrist and have it ready at hand:

- **Article no.**, **Serial no.**, **Checksum**
- **Correction value A4**, **Correction value A5**, **Correction value A6**



Typ	Type	Type	ZH 210 / 240
Artikel-Nr.	Article No.	No. d'article	
Serien-Nr.	Serial No.	No. de Série	
Prüfsumme	Checksum	Somme de contrôle	
Korrekturwert A4	Offset A4	Valeur de correction A4	
Korrekturwert A5	Offset A5	Valeur de correction A5	
Korrekturwert A6	Offset A6	Valeur de correction A6	
Meldungs-Nr.	Report-No.	No. de rapport	200070462

Fig. 5-43: In-line wrist rating plate: example in German, English, French

Precondition

- User rights: Function group **Mastering**
- T1 mode
- No program is selected.

Procedure

1. In the main menu, select **Start-up** > **Service** > **Replacement of the wrist**.

   The **Wrist exchange correction values** window opens. An illustration shows an identification plate with the relevant information as an example.
2. Fill the previously prepared values into the fields:
   - **Article no.**, **Serial no.**, **Checksum**
   - **Correction value A4**, **Correction value A5**, **Correction value A6**
3. Save.

   The entries in step 2 can only be saved if they are correct. If they are not correct, the **Save** button remains inactive and cannot be actuated.
4. Master A4, A5 and A6.

Backup and log

Backup:

On saving the new MAMES values, the robot controller creates a backup of the old MAMES file under the following path:

- C:\KRC\USER\MAM_BACKUP\[*time stamp*]\

Log:

The robot controller logs the changes in the following file:

- WristChangedHistory.log, under C:\KRC\Roboter\Log

The file contains the following information:

- Version number
- Date
- Time
- Serial number and article number of the in-line wrist
- Checksum
- Old MAMES values
- New MAMES values

Import

MAMES values can be imported. An XML file is required for this. The file name contains the serial number and article number of the wrist. To ensure that the file matches the wrist actually used, the information in the file name must be compared with the actual serial number and article number of the wrist.

For further information about importing MAMES values, please contact KUKA.

5.19 Maintenance handbook

The **Maintenance handbook** functionality is available in the KUKA System Software. The maintenance handbook enables logging of the maintenance work. The logged maintenance work can be displayed in an overview.

The robot controller uses messages to indicate when maintenance is due:

- A message is generated one month before the maintenance work is due. This message can be acknowledged.
- At the end of the month, the robot controller generates a message indicating that the maintenance is now due. This message cannot be acknowledged. Additionally, LED4 on the Controller System Panel flashes (= first LED from the left in the bottom row).

  Only when the corresponding maintenance work has been logged does the robot controller reset the message and the LED stops flashing.

The controller variant “KR C4 compact” has no Controller System Panel and no flashing lights to indicate when maintenance work is due.

The due dates are determined by the maintenance intervals specified in the KUKA maintenance agreements. The intervals are counted from the initial start-up of the robot controller. The operating hours of the robot are counted.

5.19.1 Logging maintenance

Description

It is not possible to log multiple maintenance activities of the same kind on one day.

Once saved, changes can no longer be made.

Precondition

- User rights: Function group **General configuration**

Procedure

1. In the main menu, select **Start-up** > **Service** > **Maintenance handbook**. The **Maintenance handbook** window is opened.
2. Select the **Maintenance input** tab and enter the maintenance details. Entries must be made in all boxes.
3. Press **Save**. A request for confirmation is displayed.
4. If all entries are correct, answer the request for confirmation with **Yes**.

The entries are now saved. Switching to the **Maintenance overview** tab causes the maintenance to be displayed there.

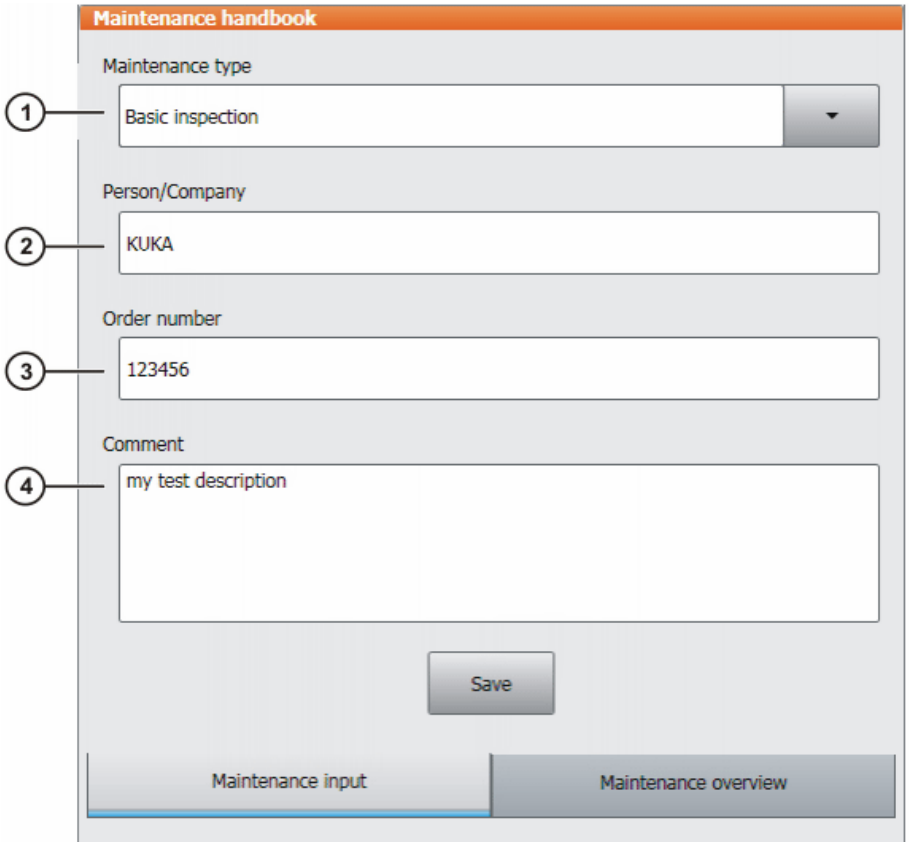


Fig. 5-44: Maintenance input

Item	Description
1	Select which type of maintenance has been carried out.
2	Enter who performed the maintenance.
3	For maintenance carried out and logged by KUKA employees: enter the order number. For other maintenance: enter any number.
4	Enter a comment.

Maintenance type

These maintenance types correspond to those in the KUKA maintenance agreements. Depending on the options used (e.g. linear axis or technology packages), other maintenance types may be available for selection.

5.19.2 Displaying a maintenance log

Description

The logged maintenance work can be displayed in an overview. If the KUKA System Software is updated, this overview is retained.

When archiving is carried out, the maintenance logs are also archived. If, when the data are restored, other maintenance work has been logged on the robot controller in the meantime, these logs are not overwritten; instead, the restored logs are added to the overview.

Procedure

1. In the main menu, select **Start-up** > **Service** > **Maintenance handbook**. The **Maintenance handbook** window is opened.
2. Select the **Maintenance overview** tab.

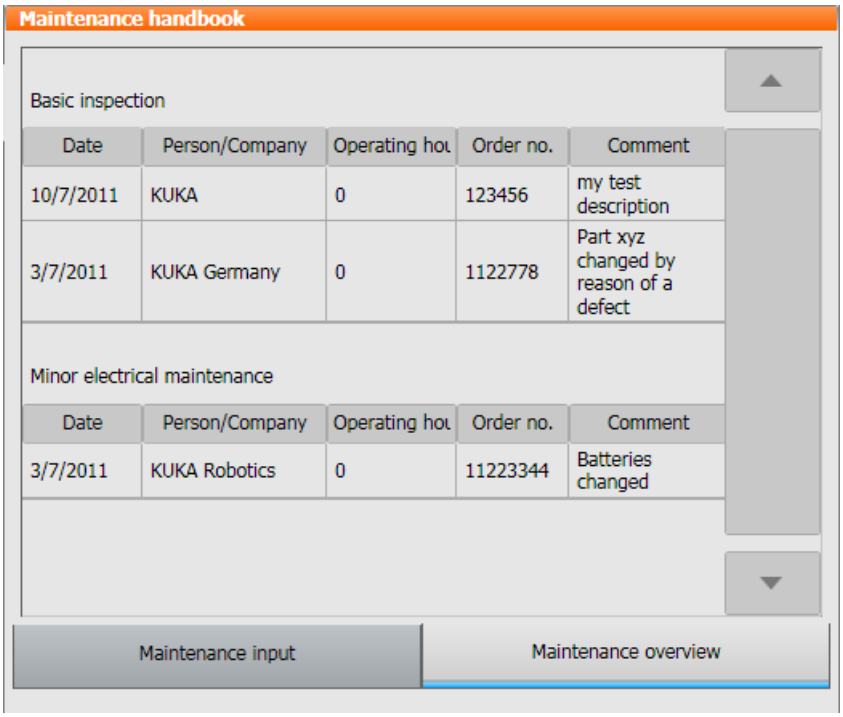


Fig. 5-45: Maintenance overview

6 Configuration

6.1 Configuring the KUKA Line Interface (KLI)

Description

The KLI is the Ethernet interface of the robot controller for external communication. It is a physical interface and can contain multiple virtual interfaces.

In order for external PCs to be able to connect to the robot controller via a network, the KLI must be configured accordingly. This is a precondition, for example, for being able to transfer WorkVisual projects to the robot controller via the KLI.

The following address ranges are used by default by the robot controller for internal purposes. IP addresses from this range cannot therefore be assigned by the user via the smartHMI. The system indicates an error.

- 169.254.0.0 … 169.254.255.255
- 172.16.0.0 … 172.16.255.255
- 172.17.0.0 … 172.17.255.255
- 192.168.0.0 … 192.168.0.255

Field buses

How the KLI has to be configured depends, among other things, on whether an Ethernet-based field bus is installed on the robot controller.

Ethernet-based field buses are:

- KR C4 PROFINET
- KR C4 EtherNet/IP

6.1.1 Configuring the Windows interface (without field bus)

Description

The Windows interface is a virtual interface of the KLI. It has a preconfigured static IP address:

- IP address: 172.31.1.147
- Subnet mask: 255.255.0.0

These values can be modified by the user. If required, it is also possible to switch to dynamic address assignment. Similarly, it is also possible to switch back from a dynamic address to a static address.

DNS Server:

If required, a DNS server address can be specified.

If the robot controller is connected to the IT network, name resolution of the devices can be carried out via the DNS server address. (In other words, the DNS server receives a query with a device name and responds with the corresponding IP address.)

“0.0.0.0” is admissible and is ignored by the system.

Precondition

- There is no Ethernet-based field bus installed on the robot controller.
- If the address type **Dynamic IP address** is to be selected: there is a DHCP server present in the network.

- User rights: function group **General configuration**
- T1 or T2 mode
- No program is selected.

Procedure

If an address or subnet field is framed in red, this indicates the presence of an error.
(>>> *6.1.5 "Error display in the Address and Subnet boxes" Page 206*)

1. In the main menu, select **Start-up** > **Network configuration**. The **Network configuration** window opens. The active Windows interface is displayed. (Default: “virtual5”)
2. Select the desired type in the **Address type:** box: **Dynamic IP address** or **Fixed IP address**

   The other types in this box (e.g. **Real-time IP address**) must not be selected.

3. Only for **Dynamic IP address**:
   - The following boxes are deactivated, as the values are automatically assigned by the DHCP server: **IP address:**, **Subnet mask:**, **Standard gateway:**
   - Fill out the **DNS Server:** box (if required).
4. Only for **Fixed IP address**: Fill out the following boxes:
   - **IP address:** Enter the IP address of the robot controller.
   - **Subnet mask:**: The selected subnet mask must match the IP network.
   - **Standard gateway:** (if required) : Specifies the IP address that can be used to leave the network.

     “0.0.0.0” is admissible and is ignored by the system.
   - **DNS Server:** (if required)
5. Press **Save**.
6. Reboot the robot controller so that the change takes effect.

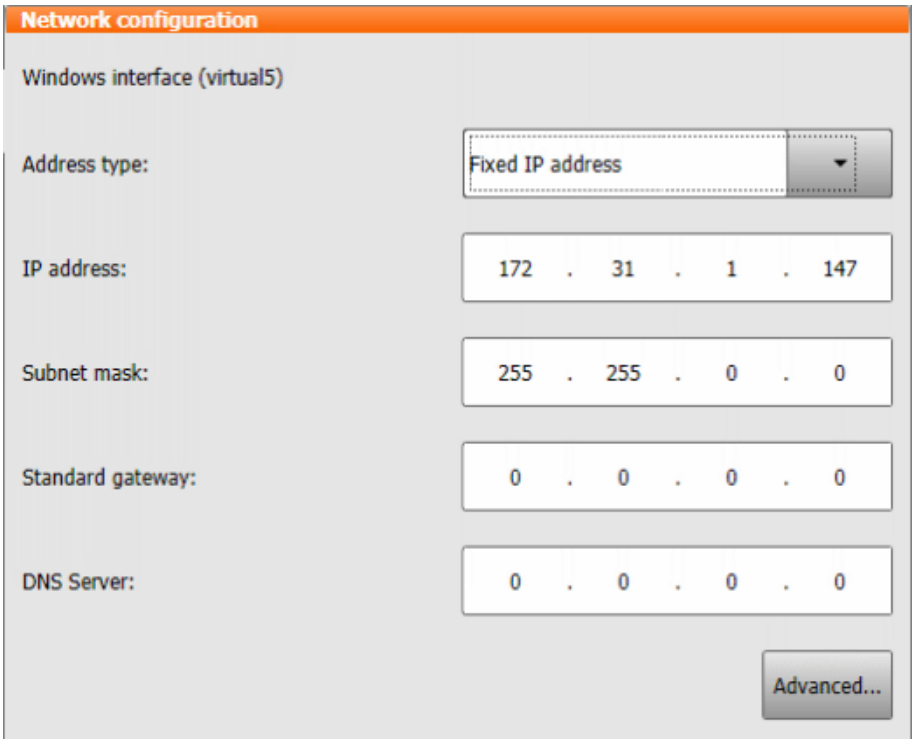


Fig. 6-1: Example: Fixed IP address

6.1.2 Configuring the field bus interface and creating the Windows interface

Description

If an Ethernet-based field bus is installed on the robot controller, it is automatically assigned virtual interface “virtual5”. This must be adapted to the field bus.

“Virtual5” can simultaneously be used as a Windows interface, but the static IP address of the field bus must then be used for Windows access.

If a different IP configuration is to be used for the Windows interface, an additional virtual interface, e.g. “virtual6”, must be added and configured for this purpose.

Precondition

- User rights: function group **Critical configurations**
- T1 or T2 mode
- No program is selected.
- If KR C4 PROFINET is installed: the PROFINET device has been named.

Information about device naming can be found in the documentation **KR C4 PROFINET**.

Procedure

If an address or subnet field is framed in red, this indicates the presence of an error.
(>>> *6.1.5 "Error display in the Address and Subnet boxes" Page 206*)

1. In the main menu, select **Start-up** > **Network configuration**. The **Network configuration** window opens. The “virtual5” interface is displayed.

2. If it has not already been selected: select the type **Fixed IP address** in the **Address type:** box.
3. Complete the boxes **IP address:** and **Subnet mask:**. Enter the address and mask that are also assigned by the PLC of the field bus.
4. Press **Advanced...**. The window for advanced network configuration opens.
5. Select the **Interfaces** tab.
6. Select the entry “virtual5” in the **Configured interfaces:** area and enter the field bus (e.g. “PROFINET”) in the **Interface designation:** box.
7. Several “Reception task” entries are displayed under the entry “virtual5”. Select the bottom one.
8. The **Reception filter:** box indicates **Accept all**. Change the entry to **Target IP address**.

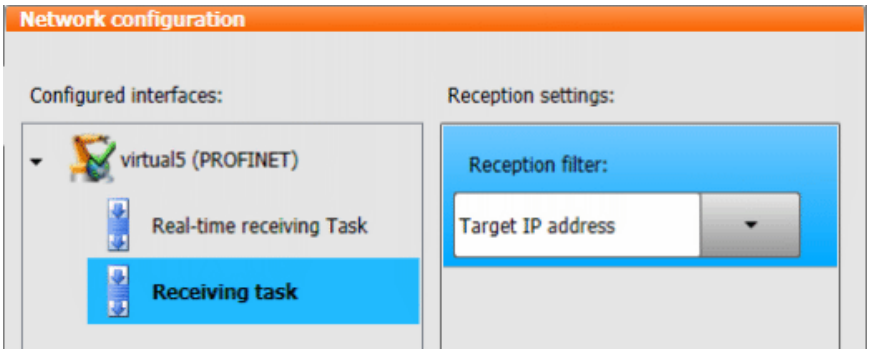


Fig. 6-2: Reception filter, with “PROFINET” as example

9. Select the entry “virtual5” and press the **Add interface** button. The entry “virtual6” is automatically created.
10. Select the entry “virtual6” and enter “WINDOWS” in the **Interface designation:** box.
11. Select the **Dynamic IP address** type in the **Address type:** box.

> **Fixed IP address** can also be selected here if a static IP address is desired. The static address must be in a different address range, however, than the address of “virtual5”.

12. Set the check mark in the check box **Windows interface**.

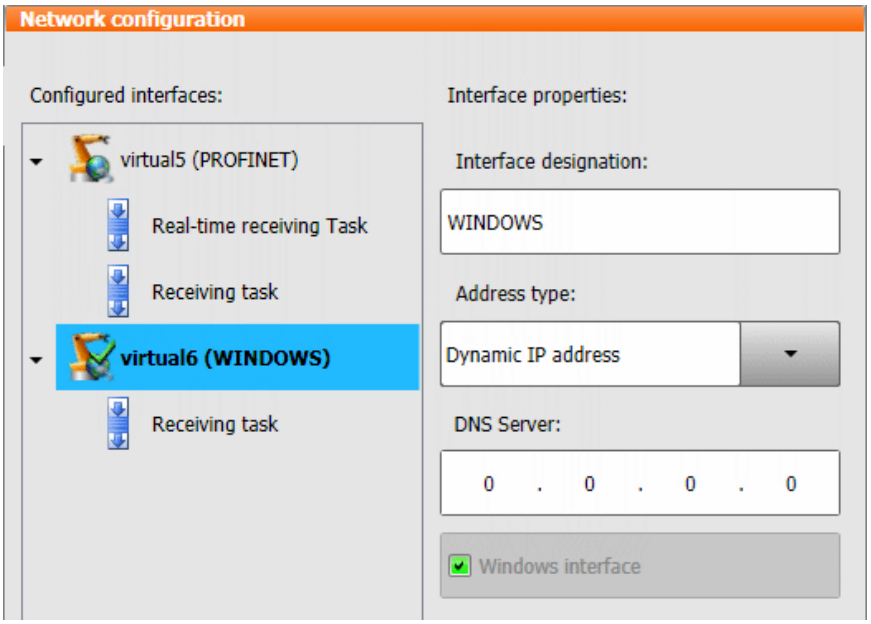


Fig. 6-3: “virtual6” interface, with “PROFINET” as example

13. Select the “Reception task” entry under the “virtual6” entry.
14. If not already the case, change the **Reception filter:** box to **Accept all**.
15. Press **Save**.
16. Close the **Network configuration** window using the Close icon.
17. Reboot the robot controller. For this, select **Shutdown** in the main menu and select the option **Reload files**.

The check mark in the **Windows interface** check box cannot be removed. Deselection is only possible by defining a different interface as the Windows interface.

6.1.3 Displaying ports of the Windows interface or enabling an additional port

It is not generally necessary to enable additional ports. If this is nevertheless to be done, KUKA Deutschland GmbH must be contacted beforehand.

NOTICE

The ports enabled by KUKA by default must not be removed. Doing so may result in the loss of functions of the robot controller.

Precondition

- User rights:
  - For display: function group **General configuration**
  - If changes are to be carried out:**Critical configurations**
- T1 or T2 mode
- No program is selected.

Procedure

1. In the main menu, select **Start-up** > **Network configuration**. The **Network configuration** window opens.

2. Press **Advanced...**. The window for advanced network configuration opens.
3. Select the **NAT** tab. A list of all the enabled ports of the Windows interface is displayed in the **Available ports:** area.
4. If a port is to be enabled:
   a. Press **Add port**. A new port with the number “0” is added to the list.
   b. Complete the boxes **Port number:** and **Permitted protocols:**.
   c. Press **Save**.

   A maximum total of 40 ports can be enabled.
5. Close the **Network configuration** window using the Close icon.
6. If modifications have been made: Reboot the robot controller with the setting **Reload files**.

6.1.4 Displaying or modifying filters

It is not generally necessary to modify the filters. If this is nevertheless to be done, KUKA Deutschland GmbH must be contacted beforehand.

NOTICE

The filters set by KUKA by default must not be modified or removed. Doing so may result in the loss of functions of the robot controller.

Precondition

- User rights:
  – For display: function group **General configuration**
  – If changes are to be carried out:**Critical configurations**
- T1 or T2 mode
- No program is selected.

Procedure

1. In the main menu, select **Start-up** > **Network configuration**. The **Network configuration** window opens.
2. Press **Advanced...**. The window for advanced network configuration opens.
3. Select the **User-defined filters** tab. The filters of the KUKA Line Interface and their properties are displayed here.
4. If modifications are required: Carry these out and press **Save**.
5. Close the **Network configuration** window using the Close icon.
6. If modifications have been made: Reboot the robot controller with the setting **Reload files**.

6.1.5 Error display in the Address and Subnet boxes

Description

The Address and Subnet boxes may be displayed with a red frame. This indicates an error and may have the following causes:

Cause	Possible remedy
The entry does not conform to the IP system. Example: The number ranges of the IP address and subnet do not match.	Check the box with the red frame and correct it. The red frame disappears.
This address is already used in one of the internal subnets. In this case, both the address and the corresponding box on the **Internal subnets** tab are displayed with a red frame.	• Change the actual address. • Or, only after consultation with KUKA: change the address of the internal subnet. The red frame disappears.

NOTICE

The subnet configuration of the robot controller may be modified only in consultation with KUKA Deutschland GmbH. Modifications carried out without consultation may result in the loss of functions of the robot controller.

Example

Example of an entry that is not system-compliant:

In the binary display, subnet masks may only contain closed groups of leading ones.

- The subnet mask “255.255.208.0” is entered.
- The box is now displayed with a red frame.

  Reason: the binary representation of “208” corresponds to “11010000” and is thus not a valid entry.
- Possible remedy: Enter 255.255.240.0.

  The binary representation of “240” corresponds to “11110000” and is thus a valid entry.

6.2 Displaying the subnet configuration of the robot controller

Description

The subnet configuration of the robot controller can be displayed. This makes it possible to compare it with the subnets of the customer network.

NOTICE

The subnet configuration of the robot controller may be modified only in consultation with KUKA Deutschland GmbH. Modifications carried out without consultation may result in the loss of functions of the robot controller.

Precondition

- T1 or T2 mode
- No program is selected.
- To modify the configuration:

  User rights: Function group **Critical configurations**

Procedure

1. In the main menu, select **Start-up** > **Network configuration**. The **Network configuration** window opens.

2. Select the **Internal subnets** tab. The subnet configuration of the robot controller is displayed.

   Only the network address is displayed. The range containing the address of the device is indicated by an “x”.

> If an address or subnet field is framed in red, this indicates the presence of an error.
> (>>> *6.1.5 "Error display in the Address and Subnet boxes" Page 206*)

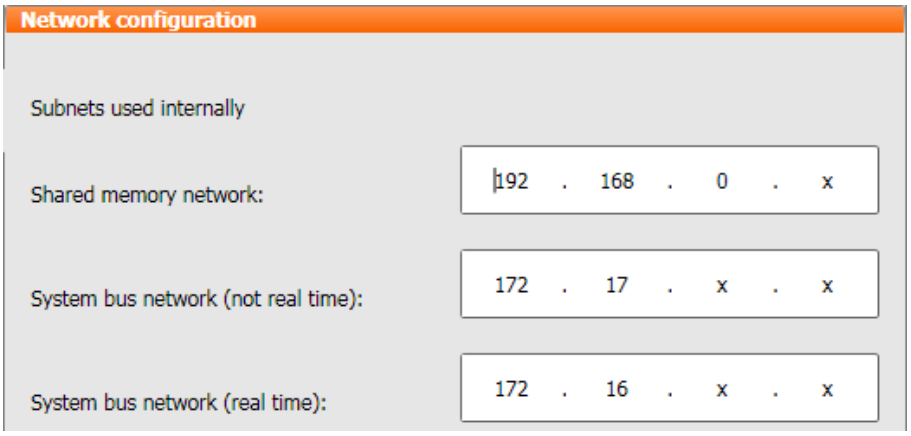


Fig. 6-4: Example: Internal subnets

6.3 Reconfiguring the I/O driver

Description

The command **I/O drivers** > **Reconfigure** causes all files in the directory C:\KRC\ROBOTER\Config\User to be reloaded. Changes made in these files are applied.

Precondition

- User rights: Function group **General configuration**
- T1 or T2 mode
- No program is selected.

> - During reconfiguration, all outputs are briefly set to zero before returning to their original state.
> - The operating mode following reconfiguration is T1.

Procedure

1. In the main menu, select **Configuration** > **Inputs/outputs** > **I/O drivers**.
2. Press the **Reconfigure** button.

   It makes no difference whether the **State** or **Configuration** tab is selected.
3. Answer the request for confirmation *Do you really want to reconfigure all I/O drivers?* with **Yes**.

   The message *Reconfiguration in progress ...* is displayed. When the message disappears, reconfiguration is completed.

6.4 Configuring safe axis monitoring functions

WARNING

Only modify the default values if it is necessary to do so. The system integrator must check whether and to what extent the values need to be modified in each specific application. He must also check whether the modification makes additional safety measures necessary, e.g. installation of a gate lock.

Preparation

The safety controller also monitors axes which are configured as couplable or grouped together in coupling groups.

- To be able to modify the parameters for these axes, the coupling must be temporarily canceled.

The coupling of the axes can be canceled and set again in WorkVisual. Information about this can be found in the WorkVisual documentation.

Precondition

- User group “Safety recovery” or higher
- T1 or T2 mode

Procedure

When saving the configuration data, a reconfiguration is automatically carried out. During reconfiguration, all outputs are briefly set to zero before returning to their original state.

1. In the main menu, select **Configuration** > **Safety configuration**.
   The **Safety configuration** window opens.
2. Select the **Axis monitoring** tab.
3. Edit the parameters as required and press **Save**.
4. Answer the request for confirmation with **Yes**. The controller is reconfigured.
5. Once the reconfiguration has been completed, the following message is displayed: *The changes were saved successfully.*
   Confirm the message with **OK**.

WARNING

Following modifications to the safety configuration, the values for the safe axis monitoring functions must be checked.
(>>> *6.5 "Checking the values for the safe axis monitoring functions" Page 214*)

WARNING

Following modifications to the **Maximum velocity T1** parameter, the new value must be checked. The new value must also be checked if it is smaller than the previous value.
(>>> *6.4.2 "Checking the limits for the maximum axis velocity in T1 mode" Page 213*)

Editable parameters

The following parameters can be set for each axis. It is not generally necessary to change the default values, however.

Parameter	Description
Braking time	Duration of the axis-specific braking ramp monitoring for safety stop 1 and safety stop 2 Default: 1,500 ms (>>> *6.4.1 "Parameter: Braking time" Page 210*)
Maximum velocity T1	Maximum velocity in T1 • Rotational axes: **1.00°… 100.00°/s** Default: 30°/s • Linear axes: **1.00 … 1,500.00 mm/s** Default: 250 mm/s This parameter enables a servo gun, for example, to be calibrated in T1 with a higher velocity than 250 mm/s. **Note:** The Cartesian velocities at the flange and at the TCP are monitored independently of this parameter and cannot exceed 250 mm/s. (>>> *6.4.2 "Checking the limits for the maximum axis velocity in T1 mode" Page 213*)
Position tolerance	Tolerance for standstill monitoring in the case of safe operational stop. The axis may still move within this tolerance when a safe operational stop is active. • Rotational axes: **0.001 … 1°** Default: 0.01° • Linear axes: **0.003 … 3 mm** Default: 0.1 mm

6.4.1 Parameter: Braking time

Description

If a safety stop 1 or 2 occurs, the safety controller monitors the braking process. Among other things, it monitors whether the axis-specific velocity remains below its monitoring ramp. If the velocity is too high, i.e. if the ramp is violated, then the safety controller triggers a safety stop 0.

The ramp monitored in T1 mode cannot be changed and applies for the braking process from the configured maximum T1 velocity of the respective axis down to standstill.

The monitoring ramp valid in the other operating modes can be adapted using the **Braking time** parameter.

The parameter **Braking time** modifies the monitoring ramp. It does not modify the actual time required by the kinematic system for braking.

WARNING

Only increase the default time if it is necessary to do so. This might be required, for example, in the case of very heavy machines and/or very heavy loads, as these cannot stop within the default time.
The safety recovery technician or safety maintenance technician must check whether and to what extent the **Braking time** value needs to be modified in each specific application. He must also check whether the modification makes additional safety measures necessary, e.g. installation of a gate lock.

The monitoring ramp (for all modes apart from T1) is determined as follows:

- The ramp starts at 106% of the rated speed of the axis. This value remains constant for the first 300 ms.
- Similarly, a constant value of 10.6% of the rated speed of the axis applies for the last 300 ms of the configured braking time.
- Over the intervening time, the permissible velocity is reduced linearly from 106% to 10.6% of the rated speed of the axis.
- Allowance for the brake closing time for safety stop 1:

  200 ms before the configured braking time elapses, the brake is commanded to close (SBC) and the drives enable signal (AF) is canceled.
- In the event of a safety stop 2, the standstill monitoring is activated after completion of the braking process, but no later than after the configured braking time.

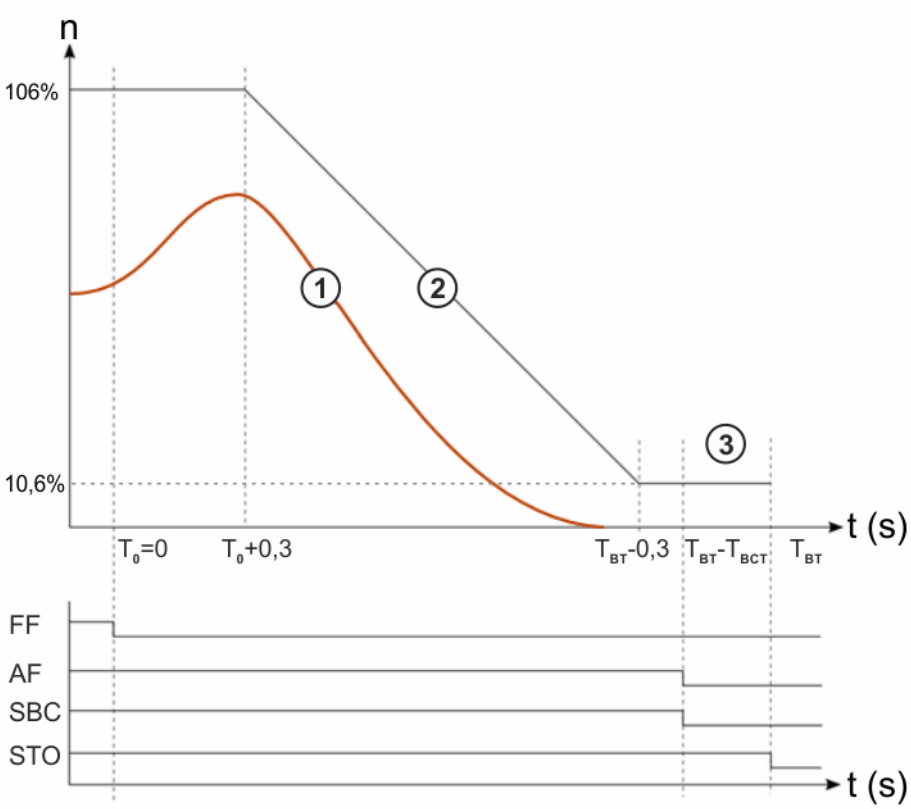


Fig. 6-5: Monitoring ramp for safety stop 1

1	Velocity profile during braking (example)
2	Monitoring ramp for braking time T_{BT}
3	Brake closing time T_{BCT} is taken into account within the monitoring ramp

n	Percentage of the rated speed of the axis
t (s)	Time (in seconds)
T_0	The moment when safety stop 1 or 2 is initiated
T_{BT}	Braking time Default-value Parameter **Braking time**: 1.5 s
T_{BCT}	Brake closing time: 0.2 s

Signals:

FF	Motion enable
AF	Drives enable
SBC	Safe Brake Control
STO	Safe Torque Off

Limitations

Braking time can be configured separately for each axis. At the moment of braking, however, the value used for all axes is always the highest value entered.

Recommendation: for greater transparency, enter the same value for all axes.

Value increased

If the value Braking time is increased, this has the following consequences:

The monitoring ramp becomes longer and flatter, i.e. monitoring is now less strict. It is now less likely that the braking process will violate the ramp.

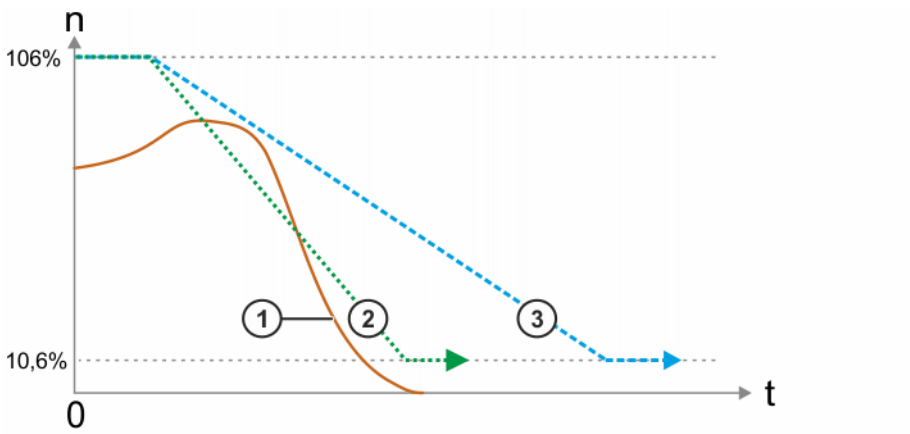


Fig. 6-6: Example: value is increased

1 Velocity profile during braking (example)
2 Monitoring (lower **Braking time** value)
3 Monitoring (higher **Braking time** value)

Value reduced

If the value "Braking time" is reduced, this has the following effect:

The monitoring ramp becomes shorter and steeper, i.e. monitoring is now stricter. There is now a higher probability that a braking process will violate the ramp.

6.4.2 Checking the limits for the maximum axis velocity in T1 mode

Description

Following modifications to the parameter **Maximum velocity T1**, the new value must be checked. The new value must also be checked if it is smaller than the previous value.

To perform the check, the value is intentionally exceeded using a test program. The safety controller then stops the robot.

Procedure

Checking the limit for rotational axes:

SAFETY INSTRUCTION
The following procedure must be followed exactly!

1. Create a test program in which the axis velocity is intentionally exceeded (e.g. by moving axis A1 at 25°/s although it is configured with 20°/s).
   a. Calculate the axis velocity $VEL_AXIS[x].
      (>>> *"Example calculation of $VEL_AXIS" Page 214*)
   b. Enter the axis velocity $VEL_AXIS[x] in the test program.
2. Execute the test program in T1 mode.
   The safety controller stops the robot.
   If the robot is stopped by the safety controller, a message with message number 15*xxx* is displayed.
3. If the robot does not stop, or if either no message or a message from a different number range is displayed, this indicates that the value for **Maximum velocity T1** has been incorrectly configured or that values have been programmed in the test program that are not appropriate for the configured maximum value.
   Check the configuration and the test program, correct if necessary and check the limit again.

Checking the limit for linear axes:

SAFETY INSTRUCTION
The following procedure must be followed exactly!

1. Create a test program in which the axis velocity is intentionally exceeded (e.g. by moving a linear axis at 110 mm/s although it is configured with 100 mm/s).
2. Execute the test program in T1 mode.
   The safety controller stops the robot.
   If the robot is stopped by the safety controller, a message with message number 15*xxx* is displayed.
3. If the robot does not stop, or if either no message or a message from a different number range is displayed, this indicates that the value for **Maximum velocity T1** has been incorrectly configured or that values have been programmed in the test program that are not appropriate for the configured maximum value.
   Check the configuration and the test program, correct if necessary and check the limit again.

Example calculation of $VEL_AXIS

Calculate the axis velocity $VEL_AXIS[x] as follows:

$VEL_AXIS[x] = ($V_{Test}$ / V_{max}) * 100 = (25°/s / 360°/s) * 100 = 7

Element	Description
x	Number of the axis
V_{test}	Desired test velocity (in this example, 25°/s) Unit: °/s
V_{max}	Maximum axis velocity according to the data sheet of the robot Unit: °/s

Enter the calculated axis velocity $VEL_AXIS[x] in the test program:

```
...
PTP {A1 -30}
HALT
$VEL_AXIS[1] = 7
PTP {A1 30}
...
```

6.5 Checking the values for the safe axis monitoring functions

Description

When the safety configuration is saved, random errors can occur in the system, resulting in the safety configuration ultimately containing values that differ from those programmed by the user. This is an exceptional occurrence, but cannot be ruled out entirely.

To rule out the possibility of such an error occurring for the parameters **Braking time** and **Position tolerance**, the values of these parameters must be verified in the diagnostic monitor. No other type of verification is possible for these parameters.

WARNING

The values must always be checked if the checksum has changed on the **Common** tab in the **Safety configuration** window, i.e. not only if the values themselves have been changed, but if any changes have been made that affect the safety configuration.
If this check is not carried out, the safety configuration may contain incorrect data. Death to persons, severe injuries or considerable damage to property may result.

Precondition

- The values most recently saved for the parameters **Braking time** and **Position tolerance** are known.

  The most recently saved values can generally be found in a checklist, sign-off sheet, or similar.

Procedure

SAFETY INSTRUCTION

The following procedure must be followed exactly!

1. In the main menu, select **Diagnosis** > **Diagnostic monitor**.

The **Diagnostic monitor** window opens.

2. Select the **Safety controller (HnfHlp)** area in the **Module** box.

   Data are now displayed for this area.
3. Compare the values displayed for **Braking time** and **Position tolerance** with the most recently saved values.
4. Result:
   - If the values match: OK.

     Close the **Diagnostic monitor** window. No further action is necessary.
   - If the values do not match:

     Enter and save the values again. If necessary, transfer the WorkVisual project to the robot controller again.

     Then carry out the check again. KUKA must be contacted if the values still do not match.

6.6 Checking the safety configuration of the robot controller

Description

The safety configuration of the robot controller must be checked in the following cases:

- After activation of a WorkVisual project on the robot controller
- Generally after changes to the machine data (independent of WorkVisual).

WARNING
If the safety configuration is not checked and updated where necessary, it may contain incorrect data. Death, injuries or damage to property may result.

Precondition

- User group “Safety recovery” or higher

Procedure

SAFETY INSTRUCTION
The following procedure must be followed exactly!

1. In the main menu, select **Configuration** > **Safety configuration**.
2. The safety configuration checks whether there are any relevant deviations between the data in the robot controller and those in the safety controller.
3. The following situations can now occur:
   a. If there are no deviations, the **Safety configuration** window is opened. No message is displayed. No further action is necessary.
   b. If there are deviations regarding the machine data, a dialog message is displayed. The deviations can now be synchronized or left unsynchronized. In either case, safety-relevant measures must be taken into consideration.

      (>>> *"Deviations" Page 216*)
   c. The safety configuration also checks whether there are any other deviations (other than in the machine data) between the robot controller and the safety controller.

If so, the **Troubleshooting wizard** window is opened. A description of the problem and a list of possible causes is displayed. The user can select the applicable cause. The wizard then suggests a solution.

Deviations

The dialog message indicates which machine data in the robot controller deviate from those in the safety controller.

The message asks whether the safety configuration is to be updated, i.e. whether the machine data of the robot controller are to be applied to the safety configuration.

- If so: Answer the query with **Yes**.

WARNING
If deviations are applied, the safety measures for start-up and recommissioning must then be carried out.

- If not: Answer the query with **No**.

WARNING
In this case, the robot must not be operated. The machine data must be checked and corrected so that either the safety configuration detects no further deviations or the detected deviations can be applied.

6.7 Checksum of the safety configuration

Description

The checksum is updated each time the safety configuration is saved. The robot controller displays the checksum in the following locations:

- In the **Safety configuration** window on the **Common** tab :

  The window can be opened from the main menu by selecting **Configuration** > **Safety configuration**.
- For user group Expert or higher:

  In the file SCTLCRC.XML in the directory C:\KRC\ROBOTER\Config\User\Common

The SCTLCRC.XML file is available so that, if required, the system integrator can read the checksum there via the higher-level controller.

During start-up and recommissioning of the industrial robot, it is advisable to check that the value is correctly read from the SCTLCRC.xml file and transferred to the higher-level controller.

SCTLCRC.XML

```
1  <?xml version="1.0" encoding="utf-8"?>

2  <SafetyInfo ParameterCrc="3702332E" />
```

Fig. 6-7: Example: Checksum in the SCTLCRC.XML file

6.8 Exporting the safety configuration (XML export)

Description

Parts of the safety configuration can be exported. The export creates an XML file. This contains only those parameters which are relevant for the safety options, e.g. SafeOperation.

- Exporting is always possible, irrespective of whether a safety option is installed or not. However, an export only makes sense if a safety option is installed.
- If no safety option is installed on the robot controller, the parameters in the XML file are filled with default values (often “0”).

In addition to exporting, it is also possible to import a safety configuration when a safety option is installed. More detailed information about exporting and importing can be found in the safety option documentation.
It is also possible to import or export safety configurations in WorkVisual. Information about this can be found in the WorkVisual documentation.

Procedure

1. In the main menu, select **Configuration** > **Safety configuration**.
   The **Safety configuration** window opens.
2. Press **Export**. The available drives are displayed.
3. Select the desired file path and press **Export**.
   The safety configuration is saved in an XML file. The file name is generated automatically.

6.9 Configuring the variable overview

This is where the variables to be displayed in the variable overview and the number of groups are defined. A maximum of 20 groups is possible. A maximum of 25 variables per group is possible. System variables and user-defined variables can be displayed.

Precondition

- User rights: Function group **General configuration**

Procedure

1. In the main menu, select **Display** > **Variable** > **Overview** > **Configuration**.
   The **Variable overview – Configuration** window is opened.
2. Make the desired settings. To edit a cell, select it.
3. Press **OK** to save the configuration and close the window.

Description

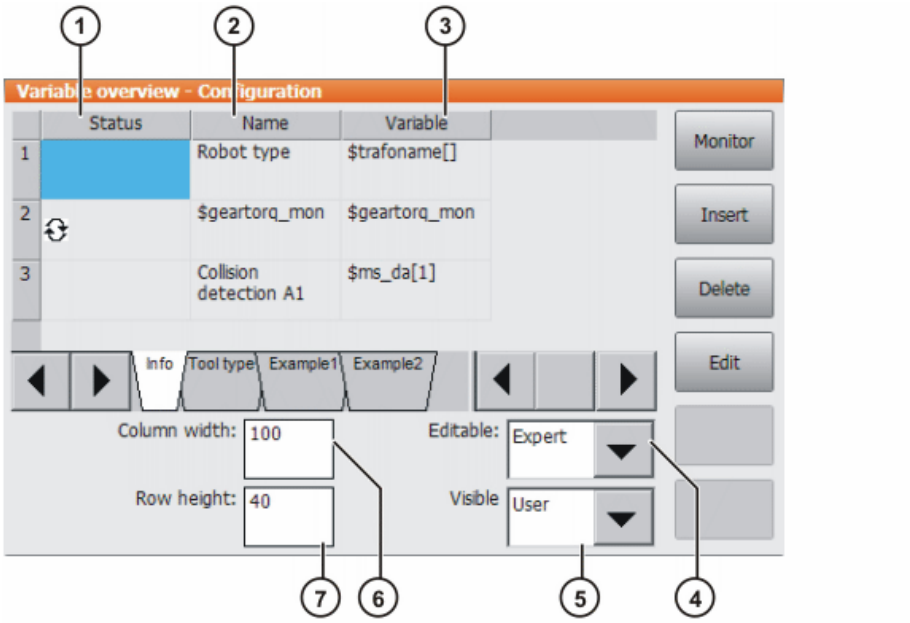


Fig. 6-8: Variable overview - Configuration

Item	Description
1	• Arrow symbol: If the value of the variable changes, the display is automatically refreshed. • No arrow symbol: The display is not automatically refreshed.
2	Descriptive name
3	Path and name of the variable **Note:** For system variables, the name is sufficient. Other variables must be specified as follows: /R1/*Program name*/*Variable name* Do not specify a folder between /R1/ and the program name. Do not add a file extension to the file name.
4	Lowest user group in which the current group can be modified.
5	Lowest user group in which the current group can be displayed.
6	Width of the selected column in mm. Enter the desired value and confirm with the Enter key.
7	Height of the selected row in mm. Enter the desired value and confirm with the Enter key.

Button	Description
Display	Switches to the variable overview. (>>> *4.19.7 "Displaying the variable overview and modifying variables" Page 105*)
Paste	Displays additional buttons: • **Row above**: Inserts a new row above the one currently selected. • **Row below**: Inserts a new row below the one currently selected. • **Group before**: Inserts a new group to the left of the one currently selected. • **Group after**: Inserts a new group to the right of the one currently selected.
Delete	Displays additional buttons: • **Line**: The selected row is deleted. • **Group**: The current group is deleted.

6.10 Changing the password

Procedure

1. Touch the **User group** icon on the smartHMI.
   or: In the main menu, select **Configuration** > **User group**.
   The user group selection window opens.
2. Select the user group for which the password is to be changed.
3. Press **Password ...** in the button bar at the bottom.
4. Enter the old password. Enter the new password twice.
   For security reasons, the entries are displayed encrypted. Upper and lower case are distinguished.
5. Press **OK**. The new password is valid immediately.

6.11 Displaying and modifying user rights

Description

Most functions of the system software are assigned to so-called function groups. In the rights management, the minimum user group required to execute the corresponding functions is defined for each of these function groups.

- The administrator can assign other user groups to the function groups.
- Every user can display which user group is currently assigned to which function group.
- The assignment of the individual functions to the function groups cannot be changed.

Information about which functions belong to which function group can be found below.
(>>> *6.11.2 "Overview of function groups" Page 221*)

Example:

The “Set jog override” function belongs to the function group **General jog settings**. This assignment cannot be changed.

As standard, the user group **Operator** is assigned to the function group **General jog settings**:

- Users of the user group **Operator** or higher thus have the right to set the jog override.
- The administrator can change this assignment.

Precondition

To modify rights:

- Administrator user group

Procedure

1. In the main menu, select **Start-up** > **Rights management**.

   The **Rights management** window opens.
2. Select the **Function groups** tab.
3. To change the right for a function group, select the desired user group in the selection box next to the group.

   Or, if required, select **Deactivated**: This setting disables the function group for all users.
4. Touch the **Close** icon. A request for confirmation is displayed, asking if the changes are to be saved.
5. Answer the query with **Yes**. The entries are saved and the window is closed.

6.11.1 Information about function groups in this documentation

One/multiple function groups

One or more function groups can be relevant for a **procedure**. They are specified under **Precondition**. In order to be able to execute the procedure, the user must belong to the highest user group that is covered by all of these function groups together.

- If only one function group is relevant for a procedure, it is specified as follows:

  "*User rights:* […]"
- If multiple function groups are relevant for a procedure, they are specified as follows:

  "*User rights of the following function groups:* […]"

Example:

The following function groups are specified under **Precondition** for the **procedure** "Move axes to pre-mastering position using the probe":

- **Mastering**
- **Jogging with the jog keys**

The user can see in the **Rights management** window which user groups are assigned to these function groups. As standard, these are:

- **Mastering**: **Expert**
- **Jogging with the jog keys**: **Operator**

The higher user group is **Expert**. The user must therefore belong to at least the user group **Expert** in order to be able to execute the procedure.

Function group + user group

One or more function groups can be relevant for a **procedure** AND additionally a permanently defined user group, e.g. **Expert**. In order to be able to execute the entire procedure, the user must belong to the highest user group that is covered by all of these specifications together.

- The additional user group is specified as follows under **Precondition** after the function groups:

  "*But at least the user group* […]"

6.11.2 Overview of function groups

The table gives an overview of which functions belong to which function groups. This assignment cannot be modified.

A detailed listing of the functions and function groups can be found in the Appendix. Information about the functions which do not belong to a function group can also be found there.

Function group	Brief description of the associated functions/ default settings
Start-up mode	**Menu item "Start-up mode" only** Default setting: **Expert**
Mastering	**All mastering/unmastering functions with the exception of dial mastering** Default setting: **Expert**
Dial mastering	**Only dial mastering** Default setting: **Administrator**
Calibration	**All calibration functions with the exception of calibration tolerances** Default setting: **Expert**
General configuration	**Most of the configurations and setup/update functions** Default setting: **Expert** **Note**: We recommend NOT assigning the user group **Administrator** to the function group.
Critical configurations	**Critical configurations, e.g. calibration tolerances, RDC data, minimize HMI or shutdown options** Default setting: **Administrator**
Diagnostic functions	**Functions under the main menu "Diagnosis" with the exception of the configurations** Default setting: **Operator**
Program execution settings	**Program control settings, e.g. program override or program run mode** Default setting: **Operator**
General jog settings	**Jog settings, e.g. jog override or current base/tool** Default user group: **Operator**
Critical jog settings	**Critical jog settings, e.g. program run mode "single step"** Default user group: **Expert**

Function group	Brief description of the associated functions/ default settings
Jogging using the 6D mouse	**Jogging using the 6D mouse** Default setting: **Operator**
Jogging with the jog keys	**Jogging with the jog keys** Default setting: **Operator**
File operations	**File operations, e.g. create new program, rename or delete** Default setting: **User**
Program selection and deselection	**Program selection and deselection** Default setting: **Operator**
Block selection	**The commands “Block selection” and “Reset program” and the “Simulate” button in wait messages** Default setting: **Operator**
General KRL program changes	**Most program changes, e.g. insert or delete commands** Default setting: **User**
Critical KRL program changes	**Critical program changes, e.g. find and replace, clean data list, or change file properties in Navigator** Default setting: **Expert**
Teach local points	**Teaching and reteaching of local points** Default setting: **User**
Teach global points	Default setting: **Deactivated**
Old motion range inline forms	**Motion commands PTP, LIN and CIRC in inline forms** Default setting: **User**
New motion range inline forms	**Motion commands SPTP, SLIN, SCIRC, SPL, as well as spline blocks and corresponding logic functions in inline forms** Default setting: **User**
Roboteam inline forms	**Inline forms for Roboteam** Default setting: **Expert**
Print functions	**Print functions for KRL programs and for the logbook** Default setting: **User**
Archive with unknown destination	**Archiving functions that offer a path selection dialog, e.g. the backup manager** Default setting: **User**
Archive to local HDD/SSD	**Archiving functions with the local HDD/SSD as destination, e.g. File\Archive\Logbook** Default setting: **User**
Archive to USB drives	**Archive functions that have USB sticks (on the cabinet or on the smartPAD) as destination** Default setting: **User**
Archive to network	**Archiving functions with the configured network path as target** Default setting: **Expert**

Function group	Brief description of the associated functions/ default settings
Partial archiving	**Archiving of individual components (applications, system data or log data). This authorization is combined with the right of the archiving destination, i.e. the higher authorization must be available in order for the function to be available.** Default setting: **Expert**
Restore	**Restoration of complete archives. Affects the menu item "Restore" and the backup manager** Default setting: **User**
Partial restoration	**Restoration of individual components (applications or system data)** Default setting: **Expert**
Restoration of critical data	**Restoration of critical data, e.g. RDC data** Default setting: **Expert**
Basic operation of the technology packages	**Basic operation of the installed technology packages, e.g. pressing a status key** Default setting: **User**
Configuration of the technology packages	**Configuration of the installed technology packages, e.g. settings in the configuration plug-in** Default setting: **Expert**
Advanced configuration of the technology packages	**Advanced configuration of the installed technology packages, particularly critical settings** Default setting: **Administrator**

6.12 Enabling or disabling operating modes for specific user groups

Description

The default assignment in the System Software defines which user group may select which operating mode.

Every user can display the current assignment. The administrator can change the assignment.

WARNING

The administrator is responsible for ensuring that safe and meaningful working is possible after changes and that the configuration takes applicable regulations into consideration.

Precondition

To modify rights:

- Administrator user group

Procedure

1. In the main menu, select **Start-up** > **Rights management**.
   The **Rights management** window opens.
2. Select the **Operating modes** tab.
   An overview indicates which user group is currently allowed to select which operating modes.

3. Activate or deactivate the check boxes as required.
4. Touch the **Close** icon. A request for confirmation is displayed, asking if the changes are to be saved.
5. Answer the query with **Yes**.

   The entries are now saved and the **Rights management** window is closed.

   The entries are automatically transferred to the smartPAD and are now visible in the connection manager.

Default assignment

The table shows the default assignment of the operating modes to the user groups.

If an operating mode is not available in a system, it is not displayed in the rights management. The display may thus vary from the table shown here.

User group	T1	T2	Aut	Ext
Operator	✓	✗	✓	✓
User	✓	✗	✓	✓
Expert	✓	✓	✓	✓
Safety recovery technician	✓	✗	✓	✓
Safety maintenance technician	✓	✓	✓	✓
Administrator	✓	✓	✓	✓

✓	The user group has the right to select this operating mode.
✗	The user group does not have the right to select this operating mode.

6.13 Energy saving mode ($ECO_LEVEL)

Description

The system variable $ECO_LEVEL can be used to operate the robot in energy saving mode. The degree of energy saving can be set to “Low”, “Middle” or “High”. Energy saving mode causes the robot axes and external axes to move more slowly. The higher the saving, the lower the velocity. How much energy is saved relative to full power depends primarily on the axis positions and cannot be predicted.

$ECO_LEVEL does not affect all motions. The following table indicates which motions it affects and which it does not:

Motion	Effect?
PTP	Yes
LIN	No
CIRC	No
CP spline motions (block and individual motion) With higher motion profile	Yes
CP spline motions (block and individual motion) Without higher motion profile	No

Motion	Effect?
PTP spline motions (block and individual motion)	Yes

If a program is reset or deselected, energy saving mode is automatically deactivated.

Energy saving mode is inactive in the following cases, even if it has been activated:

- In the case of a BCO run
- In a constant velocity range with spline
- In a time block with spline

If low values have already been programmed for acceleration and velocity, $ECO_LEVEL has little or no effect.

Depending on the robot type, the savings may be the same, or virtually the same, for both “Middle” and “High” (e.g. with a payload below 30% of the default payload).

Precondition

- $ADAP_ACC not equal to #NONE
- $OPT_MOVE not equal to #NONE

The default setting for both system variables is “not equal to #NONE”.

Syntax

`$ECO_LEVEL=`*Level*

Explanation of the syntax

Element	Description
Level	Type: ENUM • #OFF: Energy saving mode is deactivated. • #LOW: Low saving • #MIDDLE: Medium saving • #HIGH: High saving

6.14 Configuring workspaces

Workspaces can be configured for a robot. These serve to protect the system. A maximum of 8 Cartesian and 8 axis-specific workspaces can be configured at any one time. The workspaces can overlap.

Irrespective of whether it is Cartesian or axis-specific, a workspace is always of one of the two following types:

- Space that the robot must not leave

  A monitoring function is triggered if the robot leaves the space.
- Space that the robot must not enter

  A monitoring function is triggered if the robot enters the space.

Which of the two types a workspace is to belong to is defined during configuration. There are not only 2 types available for selection, however, but several more finely differentiated modes.

Exactly what reactions occur when the monitoring function is triggered also depends on the configuration.

WARNING

Workspaces serve merely to protect the system. They are not for protection of personnel. This must be provided using other means. Failure to observe this may result in death, severe injuries or damage to property.

6.14.1 Configuring Cartesian workspaces

Description

Cartesian workspaces are cuboids. The following parameters define the position and size of a Cartesian workspace:

- Origin and orientation of the workspace relative to the WORLD coordinate system
- Dimensions of the workspace, starting from the origin

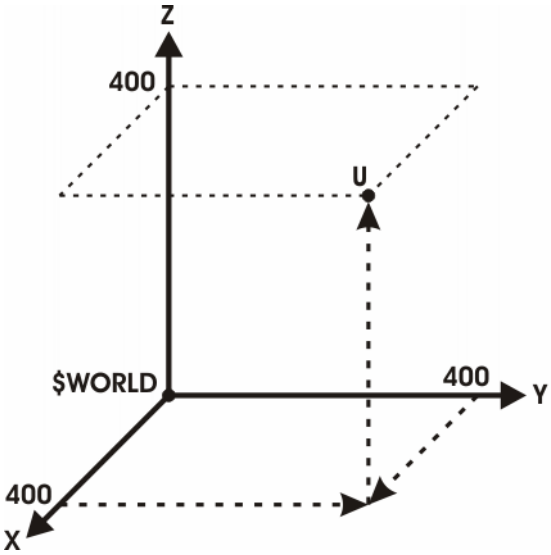


Fig. 6-9: Cartesian workspace, origin U

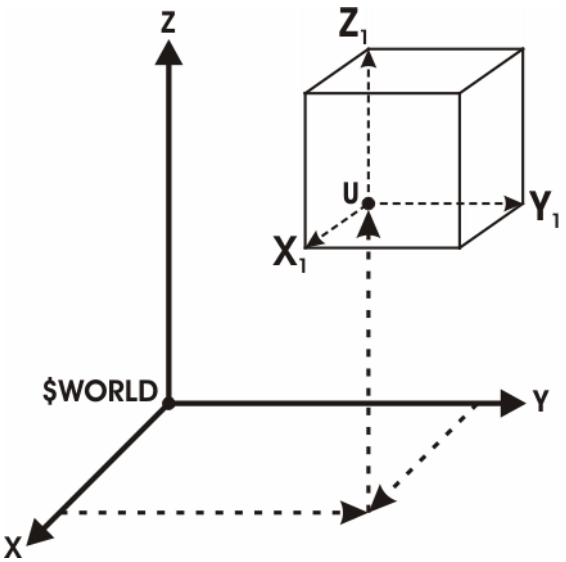


Fig. 6-10: Cartesian workspace, dimensions

Precondition

- User rights: Function group **General configuration**

- T1 or T2 mode

Procedure

1. In the main menu, select **Configuration** > **Miscellaneous** > **Workspace monitoring** > **Configuration**.

   The **Cartesian workspaces** window is opened.
2. Edit the boxes as required and press **Save**.
3. Press **Signal**. The **Signals** window is opened.
4. In the **Cartesian** group: next to the number of the workspace, enter the output that is to be set if the workspace is violated.
5. Press **Save**.
6. Close the window.

“Cartesian workspaces” window

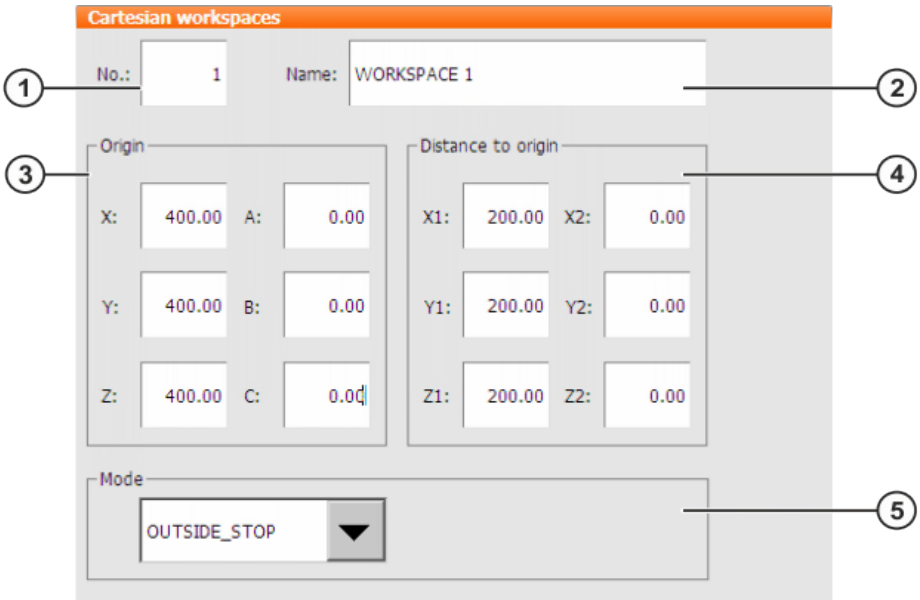


Fig. 6-11: “Cartesian workspaces” window

Item	Description
1	Number of the workspace (max. 8)
2	Designation of the workspace
3	Origin and orientation of the workspace relative to the WORLD coordinate system
4	Dimensions of the workspace in mm
5	Mode (>>> *6.14.2 "Modes for Cartesian workspaces" Page 228*)

“Signals” window

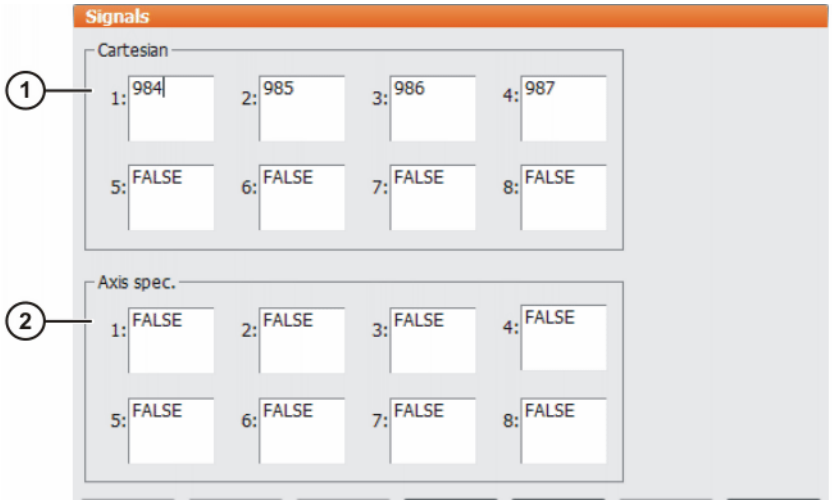


Fig. 6-12: “Signals” window

Item	Description
1	Per Cartesian workspace: Output that is set if the workspace is violated.
2	Per axis-specific workspace: Output that is set if the workspace is violated.

FALSE means that no output is assigned to this workspace.

6.14.2 Modes for Cartesian workspaces

Mode box	Description
#OFF	Workspace monitoring is deactivated.
#INSIDE	The defined output is set if the TCP or flange is located inside the workspace.
#OUTSIDE	The defined output is set if the TCP or flange is located outside the workspace.
#INSIDE_STOP	The defined output is set if the TCP, flange or wrist root point is located inside the workspace. (Wrist root point = center point of axis A5) The robot is also stopped and messages are displayed. The robot cannot be moved again until the workspace monitoring is deactivated or bypassed.
#OUTSIDE_STOP	The defined output is set if the TCP or flange is located outside the workspace. The robot is also stopped and messages are displayed. The robot cannot be moved again until the workspace monitoring is deactivated or bypassed.
#TCP_INSIDE	The defined output is set if the TCP is located inside the workspace.
#TCP_OUTSIDE	The defined output is set if the TCP is located outside the workspace.

Mode box	Description
#TCP_INSIDE_STOP	The defined output is set if the TCP is located inside the workspace. The robot is also stopped and messages are displayed. The robot cannot be moved again until the workspace monitoring is deactivated or bypassed.
#TCP_OUTSIDE_STOP	The defined output is set if the TCP is located outside the workspace. The robot is also stopped and messages are displayed. The robot cannot be moved again until the workspace monitoring is deactivated or bypassed.
(>>> *4.18 "Bypassing workspace monitoring" Page 96*)	

The TCP can only be monitored if the robot controller has information about which tool it is to refer to. The current value of $TOOL is used, or, if that is invalid, the most recent valid value.

If there has been no valid value for $TOOL since the controller was booted, the TCP cannot be monitored. The robot controller generates a corresponding message.

6.14.3 Configuring axis-specific workspaces

Description

Axis-specific workspaces can be used to further restrict the areas defined by the software limit switches in order to protect the robot, tool or workpiece.

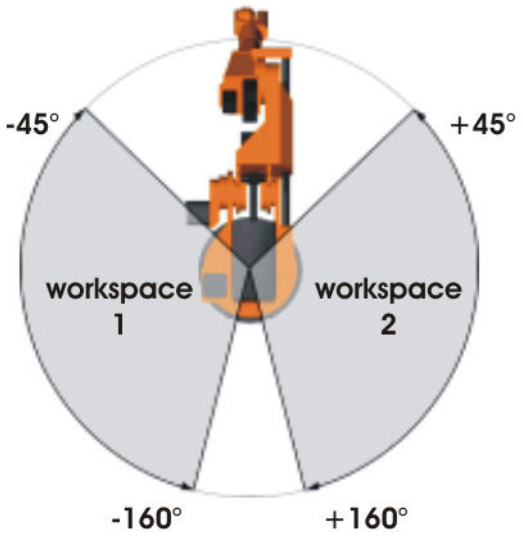


Fig. 6-13: Example of axis-specific workspaces for A1

Precondition

- User rights: Function group **General configuration**
- T1 or T2 mode

Procedure

1. In the main menu, select **Configuration** > **Miscellaneous** > **Workspace monitoring** > **Configuration**.

   The **Cartesian workspaces** window is opened.

2. Press **Axis-specific** to switch to the window **Axis-specific workspaces**.
3. Enter values, select mode and press **Save**.
4. Press **Signal**. The **Signals** window is opened.
5. In the **Axis-specific** group: next to the number of the workspace, enter the output that is to be set if the workspace is violated.
6. Press **Save**.
7. Close the window.

“Axis-specific workspaces” window

Fig. 6-14: “Axis-specific workspaces” window

Item	Description
1	Number of the workspace (max. 8)
2	Designation of the workspace
3	Lower limit for axis angle
4	Upper limit for axis angle
5	Mode (>>> *6.14.4 "Modes for axis-specific workspaces" Page 231*)

“Signals” window

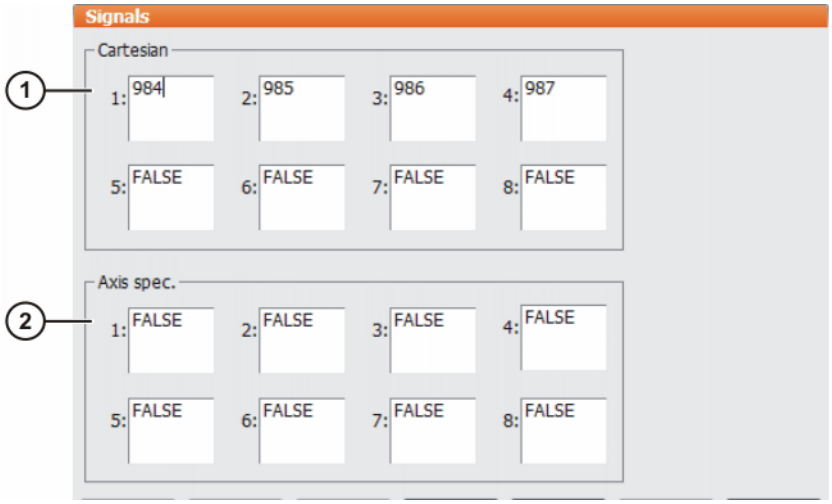


Fig. 6-15: “Signals” window

Item	Description
1	Per Cartesian workspace: Output that is set if the workspace is violated.
2	Per axis-specific workspace: Output that is set if the workspace is violated.

FALSE means that no output is assigned to this workspace.

6.14.4 Modes for axis-specific workspaces

Mode box	Description
#OFF	Workspace monitoring is deactivated.
#INSIDE	The defined output is set if the axis is located inside the workspace.
#OUTSIDE	The defined output is set if the axis is located outside the workspace.
#INSIDE_STOP	The defined output is set if the axis is located inside the workspace. The robot is also stopped and messages are displayed. The robot cannot be moved again until the workspace monitoring is deactivated or bypassed.
#OUTSIDE_STOP	The defined output is set if the axis is located outside the workspace. The robot is also stopped and messages are displayed. The robot cannot be moved again until the workspace monitoring is deactivated or bypassed.

(>>> *4.18 "Bypassing workspace monitoring" Page 96*)

6.15 Defining limits for reteaching

Description

If this function is active, local points may only be retaught within the defined limits in the user groups “User” and “Operator”. If the limits are ex-

ceeded, a message is displayed, indicating that the change is not possible.

The limits only apply to changes made using the buttons **Change** and **Touch-up** or **Cmd OK**.

In general, global points may only be modified by users with rights for the function group **Teach global points**.

Precondition

- User rights: Function group **General configuration**

Procedure

1. In the main menu, select **Configuration** > **Miscellaneous** > **Point coordinate correction limit**. The **Point coordinate correction limit** window opens.
2. Enter the desired values and activate the check box **Correction limit active**.
3. Touch the **Close** icon. A request for confirmation is displayed, asking if the changes are to be saved.
4. Answer the query with **Yes**. The entries are saved and the window is closed.

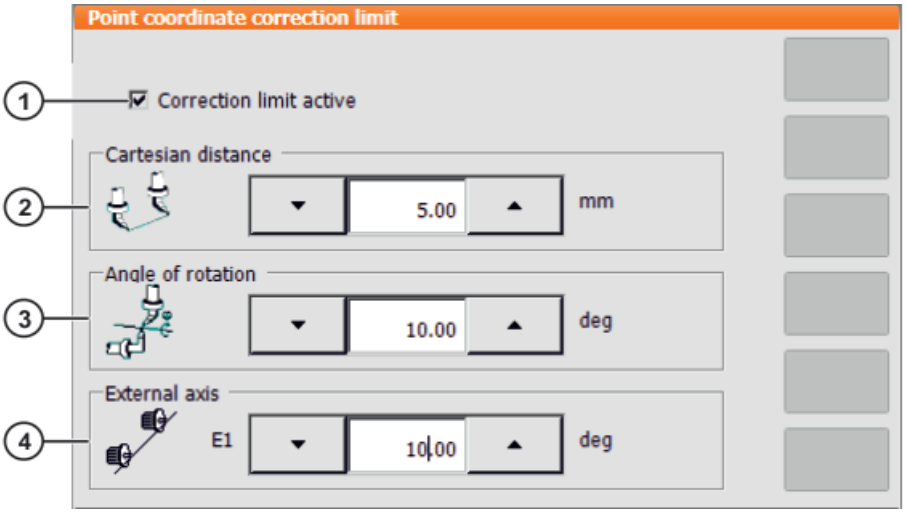


Fig. 6-16: Option window “Point coordinate correction limit”

Item	Description
1	Check box active: The limits are active. Check box not active: The limits are not active.
2	Maximum permissible change for the X, Y and Z values • **1.00 … 100.00 mm** The value represents the radius of an imagined sphere about the original point.
3	Maximum permissible change for the A, B and C values • **0.00 … 20.00 deg** Value 0.00 means: no change allowed.

Item	Description
4	This box is only displayed if at least one external axis is present. Maximum permissible change for the displayed external axis. • **0.00 … 100.00 mm**

Button	Description
External axes	The button is only available if more than one external axis is present. Displays the next external axis.

6.16 Warm-up

If a robot is started in low ambient temperatures, this results in increased friction in the gear unit. This can cause the motor current of an axis (or of more than one axis) to reach its maximum value. This stops the robot and the robot controller generates the error message *Regulator limit exceeded <axis number>*.

To avoid this, the motor current can be monitored during the warm-up phase. If a defined value is reached, the robot controller reduces the motion velocity. This, in turn, reduces the motor current.

The monitoring refers to PTP motions and PTP-CP approximate positioning blocks.
Other motions are not monitored and their velocity is not reduced. These include LIN, CIRC and all spline motions.

6.16.1 Warm-up sequence

Precondition

- $WARMUP_RED_VEL = TRUE
- Operating mode AUT or AUT EXT
- The robot is considered to be cold. This applies in the following cases:
  - Cold start
  - Or $COOLDOWN_TIME has expired.
  - Or $WARMUP_RED_VEL has been set from FALSE to TRUE.

Example

Sequence on the basis of the following example values:

```
$WARMUP_RED_VEL = TRUE
$WARMUP_TIME = 30
$COOLDOWN_TIME = 120
$WARMUP_CURR_LIMIT = 95
$WARMUP_MIN_FAC = 60
$WARMUP_SLEW_RATE = 5
```

1. The cold robot starts. The motor currents are monitored for 30 minutes ($WARMUP_TIME).
2. If the motor current of an axis exceeds 95% ($WARMUP_CURR_LIMIT) of the maximum permissible motor current, the monitoring is triggered. The robot controller then generates the message *Warm-up active* and reduces the internal override. The robot slows down and the

motor current drops. The program override displayed on the user interface remains unchanged.

The internal override is reduced to a maximum of 60% ($WARMUP_MIN_FAC) of the programmed override. There is no way of influencing how quickly the internal override is reduced.

The system variable $WARMUP_RED_FAC_ACT indicates the percentage of the program override to which the internal override is currently reduced by the warm-up:

- Internal override = $OV_PRO * $WARMUP_RED_FAC_ACT

  Example: A program override $OV_PRO of 90% and a reduction factor $WARMUP_RED_FAC_ACT of 95% result in an internal override of 90% * 95% = 85.4%.

3. Once the monitoring is no longer triggered, the robot controller increases the internal override again. This is generally the case before the minimum $WARMUP_MIN_FAC has been reached. The robot accelerates again.

   Once a second, the robot controller edges back up towards the programmed override. $WARMUP_SLEW_RATE determines the rate of increase. In the example, the internal override is increased by 5% per second.
4. It is possible that the robot is still not warm enough and that the motor current thus exceeds the maximum $WARMUP_CURR_LIMIT once again. The robot controller reacts (within $WARMUP_TIME) the same way as the first time.
5. If the robot is warm enough for the robot controller to increase the internal override all the way back up to the programmed override, the robot controller deactivates the message *Warm-up active*.
6. After 30 minutes ($WARMUP_TIME), the robot is deemed to be warmed up and the motor currents are no longer monitored.

LOG file

The following events are logged in the file “Warmup.LOG” (path “KRC:\Roboter\Log\”):

Entry	Meaning
Monitoring active	The motor currents are monitored.
Monitoring inactive	The motor currents are not monitored.
Controlling active	The velocity is reduced.
Controlling inactive	The velocity corresponds to the programmed override once again.

Example:

```
...
Date: 21.08.08 Time: 14:46:57 State: Monitoring active
Date: 21.08.08 Time: 14:54:06 State: Controlling active
Date: 21.08.08 Time: 14:54:07 State: Controlling inactive
Date: 21.08.08 Time: 18:23:43 State: Monitoring inactive
...
```

6.16.2 Configuring warm-up

The system variables for the warm-up are configured in the machine data of the robot in WorkVisual. Further information about editing machine data is contained in the WorkVisual documentation.

If one of the values is outside the permissible range, the warm-up function is not active and the velocity is not reduced.

System variable	Description
$WARMUP_RED_VEL	• **TRUE**: Warm-up function is activated. • **FALSE**: Warm-up function is deactivated. (Default) If $WARMUP_RED_VEL is set from FALSE to TRUE, this sets the runtime of the robot to zero. The robot is deemed to be cold, irrespective of how long it was previously under servo control.
$WARMUP_TIME	Time during which the motor currents are monitored by the warm-up function. If the cold robot is started, a runtime value is incremented. If the robot is not under servo control, the value is decremented. If the runtime is greater than $WARMUP_TIME, the robot is deemed to be warmed up and the motor currents are no longer monitored. • > 0.0 min
$COOLDOWN_TIME	If the warm robot is not under servo control, a standstill value is incremented. If the robot is under servo control, the value is decremented. If the standstill time is greater than $COOLDOWN_TIME, the robot is deemed to be cold and the motor currents are monitored. (Precondition: $WARMUP_RED_VEL = TRUE.) If the controller of a warm robot is shut down and restarted with a warm restart, the time the controller was switched off is counted as standstill time. • > 0.0 min
$WARMUP_CURR_LIMIT	Maximum permissible motor current during warm-up (relative to the regular maximum permissible motor current) Regular maximum permissible motor current = ($CURR_LIM * $CURR_MAX) / 100 • 0 … 100%
$WARMUP_RED_FAC_ACT	Current reduction factor for program override • $WARMUP_MIN_FAC … 100% The reduction factor indicates the percentage of the program override to which the internal override is currently reduced by the warm-up. If the warm-up function is not configured or is inactive, the reduction factor is 100%.
$WARMUP_MIN_FAC	Minimum program override reduction factor for reducing the velocity of the robot The internal override is reduced at most to the factor of the programmed override defined here. • 0 … 100%

System variable	Description
$WARMUP_SLEW_RATE	Rate of increase for the increase in velocity Once the monitoring is no longer triggered, the robot controller increases the internal override again. The rate of increase is defined here. • > 0.0%/s

6.17 Defining calibration tolerances

Only modify the default values in exceptional cases. Otherwise, increased error messages and inaccuracy may result.

Precondition

- User rights: Function group **Critical configurations**

Procedure

- In the main menu, select **Start-up** > **Calibrate** > **Tolerances**.

Description

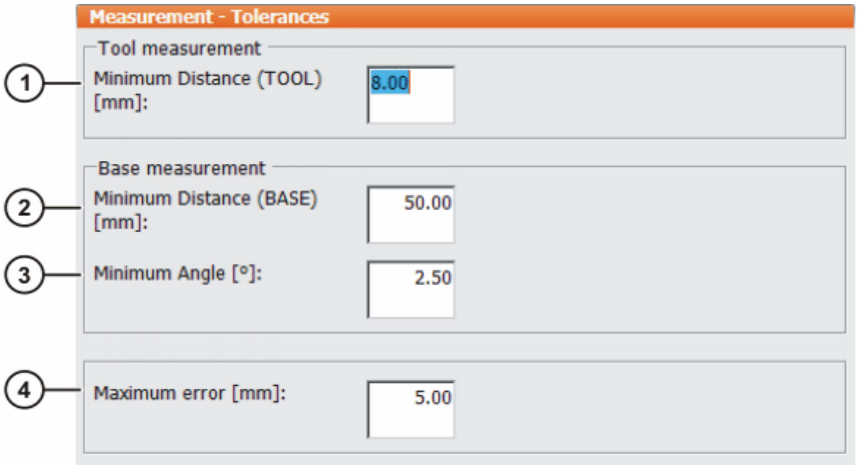


Fig. 6-17: Default error tolerances

Item	Description
1	The minimum distance for tool calibration. • **0 … 200 mm**
2	The minimum distance for base calibration. • **0 … 200 mm**
3	The minimum angle between the straight lines through the 3 calibration points in base calibration. • **0 … 360°**
4	Maximum error in calculation. • **0 … 200 mm**

The following buttons are available:

Button	Description
Default	Restores the default settings. The data must then be saved by pressing **OK**.

6.18 Configuring backward motion

Description

The configuration options described below apply to backward motion using the Start backwards key. They do not apply to other backward motion functionalities, e.g. backward motion as part of fault strategies in technology packages.

The following default settings are valid for backward motion using the Start backwards key:

- Backward motion is active.
- A maximum of 30 motions can be saved in the buffer.
- When backward motion is started, the robot controller does not indicate this by means of a message.

If other settings are desired, these must be entered in KrcConfig.XML.

Precondition

- User rights of the following function groups:
  - **Critical KRL program changes**
  - **Critical configurations**
- T1, T2 or AUT mode

Procedure

1. Open the file KrcConfig.XML in the directory C:\KRC\ROBOTER\Config\User\Common.
2. Before the last line, i.e. before `</KrcConfig>`, insert the entry BACKWARD_STEP.
3. Set the parameters to the desired values.
4. Close KrcConfig.XML. Respond to the request for confirmation asking whether the changes should be saved by pressing **Yes**.
5. Reboot the robot controller with the settings **Cold start** and **Reload files**.

BACKWARD_STEP

```
176    </KRLDIAG>

177  <BACKWARD_STEP ENABLE="true" MOVEMENTS="20"
     ↳ BACKWARD_WARNING="true"/>

178  </KrcConfig>
```

Fig. 6-18: Example: BACKWARD_STEP

The contents of the line before BACKWARD_STEP are not relevant for backward motion and may differ from this example.

If not all parameters of BACKWARD_STEP are listed, the default values are valid for those that are not listed.

If BACKWARD_STEP is removed from KrcConfig.XML again, the default values are valid for all parameters again.

Parameter	Description/default
ENABLE	Type: BOOL • **TRUE** (= default): activated, i.e. backward motion using the Start backwards key is possible. • **FALSE**: deactivated, i.e. backward motion using the Start backwards key is not possible.
MOVE-MENTS	Type: INT Maximum number of motions recorded for backward motion • **0 … 60** (default = 30)
BACK-WARD_WAR NING	Type: BOOL • **TRUE**: The robot controller generates the following message when the Start backwards key is pressed for the first time after forward motion: *Caution! Robot is moving backwards.*. The user must acknowledge the message and press the Start backwards key again. • **FALSE** (= default): No message

6.19 Configuring Automatic External

Description

If robot processes are to be controlled centrally by a higher-level controller (e.g. a PLC), this is carried out using the Automatic External interface.

The higher-level controller transmits the signals for the robot processes (e.g. fault acknowledgment, program start, etc.) to the robot controller via the Automatic External interface. The robot controller transmits information about operating states and fault states to the higher-level controller.

Overview

To enable use of the Automatic External interface, the following configurations must be carried out:

Step	Description
1	Configuration of the CELL.SRC program. (>>> *6.19.1 "Configuring CELL.SRC" Page 239*)
2	Configuration of the inputs/outputs of the Automatic External interface. (>>> *6.19.2 "Configuring Automatic External inputs/outputs" Page 240*)
3	Only if error numbers are to be transmitted to the higher-level controller: configuration of the P00.DAT file. (>>> *6.19.3 "Transmitting error numbers to the higher-level controller" Page 247*)

6.19.1 Configuring CELL.SRC

Description

In Automatic External mode, programs are called using the program CELL.SRC.

Program

```
1 DEF CELL ( )
…
6 INIT
7 BASISTECH INI
8 CHECK HOME
9 PTP HOME Vel= 100 % DEFAULT
10 AUTOEXT INI
11 LOOP
12 P00 (#EXT_PGNO,#PGNO_GET,DMY[],0 )
13 SWITCH PGNO ; Select with Programnumber
14
15 CASE 1
16 P00 (#EXT_PGNO,#PGNO_ACKN,DMY[],0 )
17 ;EXAMPLE1 ( ) ; Call User-Program
18
19 CASE 2
20 P00 (#EXT_PGNO,#PGNO_ACKN,DMY[],0 )
21 ;EXAMPLE2 ( ) ; Call User-Program
22
23 CASE 3
24 P00 (#EXT_PGNO,#PGNO_ACKN,DMY[],0 )
25 ;EXAMPLE3 ( ) ; Call User-Program
26
27 DEFAULT
28 P00 (#EXT_PGNO,#PGNO_FAULT,DMY[],0 )
29 ENDSWITCH
30 ENDLOOP
31 END
```

Line	Description
12	The robot controller calls the program number from the higher-level controller.
15	CASE branch for program number = 1
16	Receipt of program number 1 is communicated to the higher-level controller.
17	The user-defined program EXAMPLE1 is called.
27	DEFAULT = the program number is invalid.
28	Error treatment in the case of an invalid program number

Precondition

- User rights: Function group **Critical KRL program changes**

Procedure

1. Open the program CELL.SRC in the Navigator. (This program is located in the folder "R1".)

2. In the section CASE 1, replace the name EXAMPLE1 with the name of the program that is to be called via program number 1. Delete the semicolon in front of the name.

```
…
15 CASE 1
16 P00 (#EXT_PGNO,#PGNO_ACKN,DMY[],0 )
17 MY_PROGRAM ( ) ; Call User-Program
…
```

3. For all further programs, proceed as described in step 2. If required, add additional CASE branches.
4. Close the program CELL.SRC. Respond to the request for confirmation asking whether the changes should be saved with **Yes**.

6.19.2 Configuring Automatic External inputs/outputs

Procedure

1. In the main menu, select **Configuration** > **Inputs/outputs** > **Automatic External**.
2. In the **Value** column, select the cell to be edited and press **Edit**.
3. Enter the desired value and save it by pressing **OK**.
4. Repeat steps 2 and 3 for all values to be edited.
5. Close the window. The changes are saved.

Description

	Term	Type	Name	Value
1	Type programno.	Var	PGNO_TYPE	1
2	programno. reflection	Var	REFLECT_PROG_NR	0
3	Bitwidth programno.	Var	PGNO_LENGTH	8
4	First bit programno.	I/O	PGNO_FBIT	33
5	Parity bit	I/O	PGNO_PARITY	41
6	Programno. valid	I/O	PGNO_VALID	42
7	Programstart	I/O	$EXT_START	1026
8	Move enable	I/O	$MOVE_ENABLE	1025
9	Error confirmation	I/O	$CONF_MESS	1026
10	Drives off (invers)	I/O	$DRIVES_OFF	1025
11	Drives on	I/O	$DRIVES_ON	140
12	Activate interface	I/O	$I_O_ACT	1025

Fig. 6-19: Configuring Automatic External inputs

Fig. 6-20: Configuring Automatic External outputs

Item	Description
1	Number
2	Long text name of the input/output
3	Type • **Green**: Input/output • **Yellow**: variable or system variable ($...)
4	Name of the signal or variable
5	Input/output number or channel number
6	The outputs are thematically assigned to tabs.

6.19.2.1 Automatic External inputs

PGNO_TYPE

Type: Variable

This variable defines the format in which the program number sent by the higher-level controller is read.

Value	Description	Example
1	Read as binary number. The program number is transmitted by the higher-level controller as a binary coded integer.	0 0 1 0 0 1 1 1 => PGNO = 39
2	Read as BCD value. The program number is transmitted by the higher-level controller as a binary coded decimal.	0 0 1 0 0 1 1 1 => PGNO = 27

Value	Description	Example
3	Read as “1 of n”*. The program number is transmitted by the higher-level controller or the periphery as a "1 of n" coded value.	0 0 0 0 0 0 0 1 => PGNO = 1 0 0 0 0 1 0 0 0 => PGNO = 4

* When using this transmission format, the values of PGNO_REQ, PGNO_PARITY and PGNO_VALID are not evaluated and are thus of no significance.

REFLECT_PROG_NR

Type: Variable

This variable defines whether the program number is to be mirrored to an output range. The output of the signal starts with the output defined using PGNO_FBIT_REFL.

Value	Description
0	Function deactivated
1	Function activated

PGNO_LENGTH

Type: Variable

This variable determines the number of bits in the program number sent by the higher-level controller. Range of values: 1 … 16.

Example: PGNO_LENGTH = 4 => the external program number is 4 bits long.

If PGNO_TYPE has the value 2, only 4, 8, 12 and 16 are permissible values for the number of bits.

PGNO_FBIT

Input representing the first bit of the program number. Range of values: 1 … 8192.

Example: PGNO_FBIT = 5 => the external program number begins with the input $IN[5].

PGNO_PARITY

Input to which the parity bit is transferred from the higher-level controller.

Input	Function
Negative value	Odd parity
0	No evaluation
Positive value	Even parity

(>>> *6.19.2.2 "Odd / even parity" Page 244*)

If PGNO_TYPE has the value 3, PGNO_PARITY is not evaluated.

PGNO_VALID

Input to which the command to read the program number is transferred from the higher-level controller.

Input	Function
Negative value	Number is transferred at the falling edge of the signal.
0	Number is transferred at the rising edge of the signal on the EXT_START line.
Positive value	Number is transferred at the rising edge of the signal.

If PGNO_TYPE has the value 3, PGNO_VALID is not evaluated.

$EXT_START

If the I/O interface is active, this input can be set to start or continue a program.

Only the rising edge of the signal is evaluated.

NOTICE

There is no BCO run in Automatic External mode. This means that the robot moves to the first programmed position after the start at the programmed (not reduced) velocity and does not stop there.

$MOVE_ENABLE

This input is used by the higher-level controller to check the robot drives.

Signal	Function
TRUE	Jogging and program execution are possible.
FALSE	All drives are stopped and all active commands inhibited.

If the drives have been switched off by the higher-level controller, the message “GENERAL MOTION ENABLE” is displayed. It is only possible to move the robot again once this message has been reset and another external start signal has been given.

During commissioning, the variable $MOVE_ENABLE is often configured with the value $IN[1025]. If a different input is not subsequently configured, no external start is possible.

$CHCK_MOVENA

Type: Variable

If the variable $CHCK_MOVENA has the value FALSE, $MOVE_ENABLE can be bypassed. The value of the variable can only be changed in the file C:\KRC\ROBOTER\KRC\STEU\Mada\$OPTION.DAT.

Signal	Function
TRUE	MOVE_ENABLE monitoring is activated.
FALSE	MOVE_ENABLE monitoring is deactivated.

In order to be able to use MOVE_ENABLE monitoring, $MOVE_ENABLE must have been configured with the input $IN[1025]. Otherwise, $CHCK_MOVENA has no effect.

$CONF_MESS

Setting this input enables the higher-level controller to acknowledge error messages automatically as soon as the cause of the error has been eliminated.

Only the rising edge of the signal is evaluated.

$DRIVES_OFF

If there is a low-level pulse of at least 20 ms duration at this input, the higher-level controller switches off the robot drives.

$DRIVES_ON

If there is a high-level pulse of at least 20 ms duration at this input, the higher-level controller switches on the robot drives.

$I_O_ACT

If this input is TRUE, the Automatic External interface is active. Default setting: $IN[1025].

6.19.2.2 Odd / even parity

Description

A parity bit is a bit which is added to a bit sequence and has a checking function. It indicates whether the bit sequence contains an even or odd sum of ones.

If the entire data block – consisting of the bit sequence and the parity bit – is transferred and the parity bit then no longer matches the sequence, this means that an error has occurred during transfer.

The meaning of the relevant bit value (“even” or “odd”) depends on the applicable parity protocol: “even parity” or “odd parity”.

Even parity

Even parity

The sum of the ones in the entire data block (bit sequence + parity bit) must be even.

Sum of ones in the bit sequence	Parity bit
Even	0
Odd	1

Example:

Bit sequence	Parity bit
0011.1010	0
1010.0100	1

Odd parity

Odd parity

The sum of the ones in the entire data block (bit sequence + parity bit) must be odd.

Sum of ones in the bit sequence	Parity bit
Even	1
Odd	0

Example:

Bit sequence	Parity bit
0011.1010	1
1010.0100	0

6.19.2.3 Automatic External outputs

$RC_RDY1

Ready for program start.

$ALARM_STOP

This output is reset in the following EMERGENCY STOP situations:

- The EMERGENCY STOP device on the smartPAD is pressed.
- External EMERGENCY STOP

In the case of an EMERGENCY STOP, the nature of the EMERGENCY STOP can be recognized from the states of the outputs **$ALARM_STOP** and **$ALARM_STOP_INTERN**:

- Both outputs are FALSE: the EMERGENCY STOP was triggered on the smartPAD.
- **$ALARM_STOP** is FALSE, **$ALARM_STOP_INTERN** is TRUE: external EMERGENCY STOP.

$USER_SAF

This output is reset if the safety fence monitoring switch is opened (AUT mode) or an enabling switch is released (T1 or T2 mode).

$PERI_RDY

By setting this output, the robot controller communicates to the higher-level controller the fact that the intermediate circuit is fully charged and that the robot drives are ready.

$ROB_CAL

The signal is FALSE as soon as a robot axis has been unmastered

$I_O_ACTCONF

This output is TRUE if Automatic External mode is selected and the input $I_O_ACT is TRUE.

$STOPMESS

This output is set by the robot controller in order to communicate to the higher-level controller any message occurring which requires the robot to be stopped. (Examples: EMERGENCY STOP, Motion enable or Operator safety)

PGNO_FBIT _REFL

Output representing the first bit of the program number. Precondition: The input REFLECT_PROG_NR has the value 1.

The size of the output area depends on the number of bits defining the program number (PGNO_LENGTH).

If a program selected by the PLC is deselected by the user, the output area starting with PGNO_FBIT_REFL is set to FALSE. In this way, the PLC can prevent a program from being restarted manually.

PGNO_FBIT_REFL is also set to FALSE if the interpreter is situated in the CELL program.

$ALARM_STOP _INTERN

Previous name: **Int. E-Stop**

This output is set to FALSE if the EMERGENCY STOP device on the smartPAD is pressed.

In the case of an EMERGENCY STOP, the nature of the EMERGENCY STOP can be recognized from the states of the outputs **$ALARM_STOP** and **$ALARM_STOP_INTERN**:

- Both outputs are FALSE: the EMERGENCY STOP was triggered on the smartPAD.
- **$ALARM_STOP** is FALSE, **$ALARM_STOP_INTERN** is TRUE: external EMERGENCY STOP.

$PRO_ACT

This output is set whenever a process is active at robot level. The process is therefore active as long as a program or an interrupt is being processed. Program processing is set to the inactive state at the end of the program only after all pulse outputs and all triggers have been processed.

In the event of an error stop, a distinction must be made between the following possibilities:

- If interrupts have been activated but not processed at the time of the error stop, the process is regarded as inactive ($PRO_ACT=FALSE).
- If interrupts have been activated and processed at the time of the error stop, the process is regarded as active ($PRO_ACT=TRUE) until the interrupt program is completed or a STOP occurs in it ($PRO_ACT=FALSE).
- If interrupts have been activated and a STOP occurs in the program, the process is regarded as inactive ($PRO_ACT=FALSE). If, after this, an interrupt condition is met, the process is regarded as active ($PRO_ACT=TRUE) until the interrupt program is completed or a STOP occurs in it ($PRO_ACT=FALSE).

PGNO_REQ

A change of signal at this output requests the higher-level controller to send a program number.

If PGNO_TYPE has the value 3, PGNO_REQ is not evaluated.

APPL_RUN

By setting this output, the robot controller communicates to the higher-level controller the fact that a program is currently being executed.

$PRO_MOVE

Means that a synchronous axis is moving, including in jog mode. The signal is thus the inverse of $ROB_STOPPED.

$IN_HOME

This output communicates to the higher-level controller whether or not the robot is in its HOME position.

$ON_PATH

This output remains set as long as the robot stays on its programmed path. The output ON_PATH is set after the BCO run. This output remains set until the robot leaves the path, the program is reset or block selection

is carried out. The ON_PATH signal has no tolerance window, however; as soon as the robot leaves the path the signal is reset.

$NEAR_POSRET

This signal allows the higher-level controller to determine whether or not the robot is situated within a sphere about the position saved in $POS_RET. The higher-level controller can use this information to decide whether or not the program may be restarted.

The user can define the radius of the sphere in the file $CUSTOM.DAT using the system variable $NEARPATHTOL.

$ROB_STOPPED

The signal is set when the robot is at a standstill. In the event of a WAIT statement, this output is set during the wait.

The signal is thus the inverse of $PRO_MOVE.

$T1, $T2, $AUT, $EXT

These outputs are set when the corresponding operating mode is selected.

6.19.3 Transmitting error numbers to the higher-level controller

Error numbers of the robot controller in the range 1 to 255 can be transmitted to the higher-level controller. To transmit the error numbers, the file P00.DAT, in the directory C:\KRC\ROBOTER\KRC\R1\TP, must be configured as follows:

```
1 DEFDAT P00
2
3 BOOL PLC_ENABLE=TRUE ; Enable error-code transmission to
plc
4 INT I
5 INT F_NO=1
6 INT MAXERR_C=1 ; maximum messages for $STOPMESS
7 INT MAXERR_A=1 ; maximum messages for APPLICATION
8 DECL STOPMESS MLD
9 SIGNAL ERR $OUT[25] TO $OUT[32]
10 BOOL FOUND
11
12 STRUC PRESET INT OUT,CHAR PKG[3],INT ERR
13 DECL PRESET P[255]
…
26 P[1]={OUT 2,PKG[] "P00",ERR 10}
…
30 P[128]={OUT 128,PKG[] "CTL",ERR 1}
…
35 STRUC ERR_MESS CHAR P[3],INT E
36 DECL ERR_MESS ERR_FILE[64]
37 ERR_FILE[1]={P[] "XXX",E 0}
…
96 ERR_FILE[64]={P[] "XXX",E 0}
97 ENDDAT
```

Line	Description
3	PLC_ENABLE must be TRUE.
6	Enter the number of controller errors for the transmission of which parameters are to be defined.
7	Enter the number of application errors for the transmission of which parameters are to be defined.
9	Specify which robot controller outputs the higher-level controller should use to read the error numbers. There must be 8 outputs.
13	In the following section, enter the parameters of the errors. P[1] … P[127]: range for application errors P[128] … P[255]: range for controller errors
26	Example of parameters for application errors: • OUT 2 = error number 2 • PKG[] "P00" = technology package • ERR 10 = error number in the selected technology package
30	Example of parameters for controller errors: • OUT 128 = error number 128 • PKG[] "CTL" = technology package • ERR 1 = error number in the selected technology package
37 … 96	The last 64 errors that have occurred are stored in the ERR_FILE memory.

6.19.4 Signal diagrams

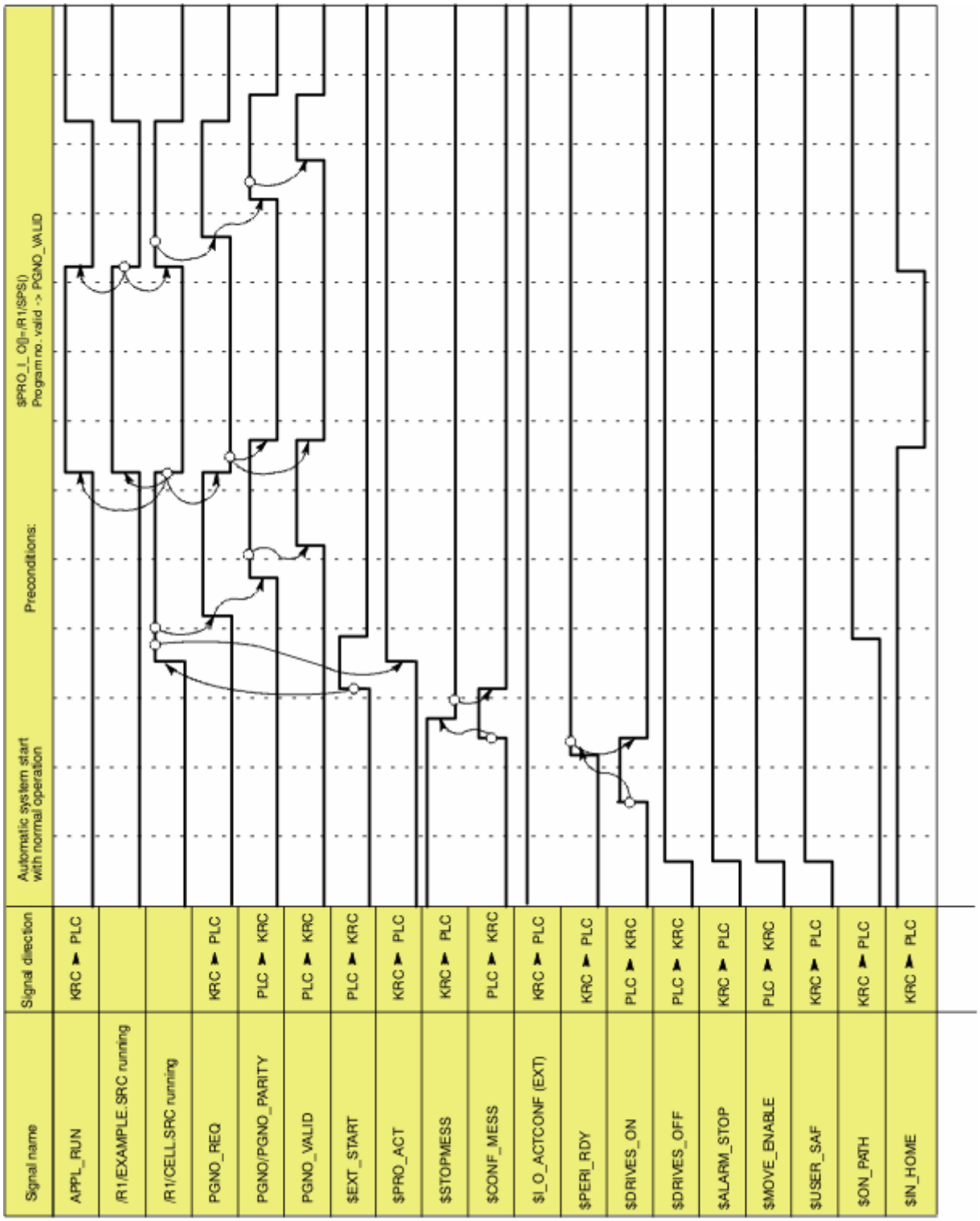


Fig. 6-21: Automatic system start and normal operation with program number acknowledgement by means of PGNO_VALID

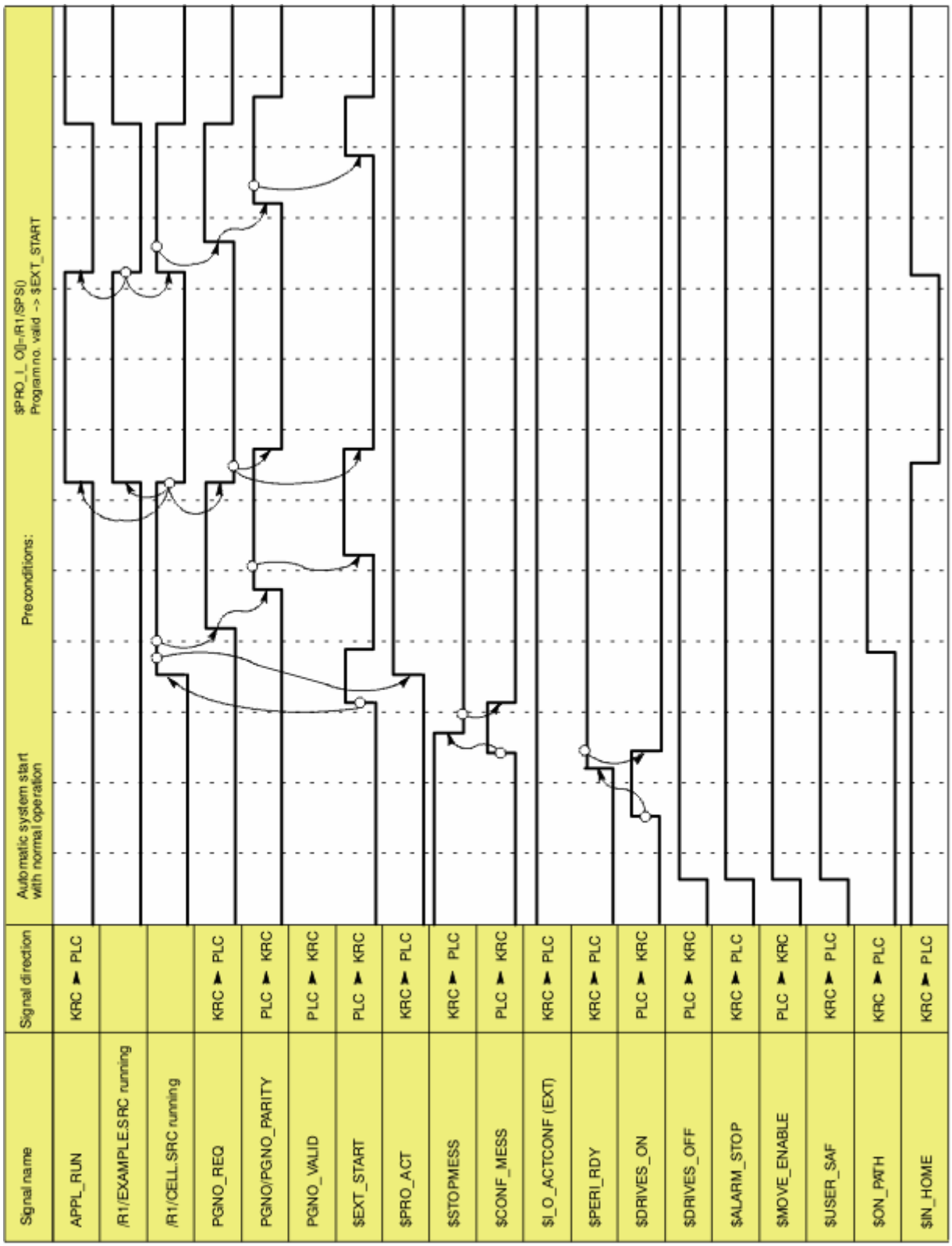


Fig. 6-22: Automatic system start and normal operation with program number acknowledgement by means of $EXT_START

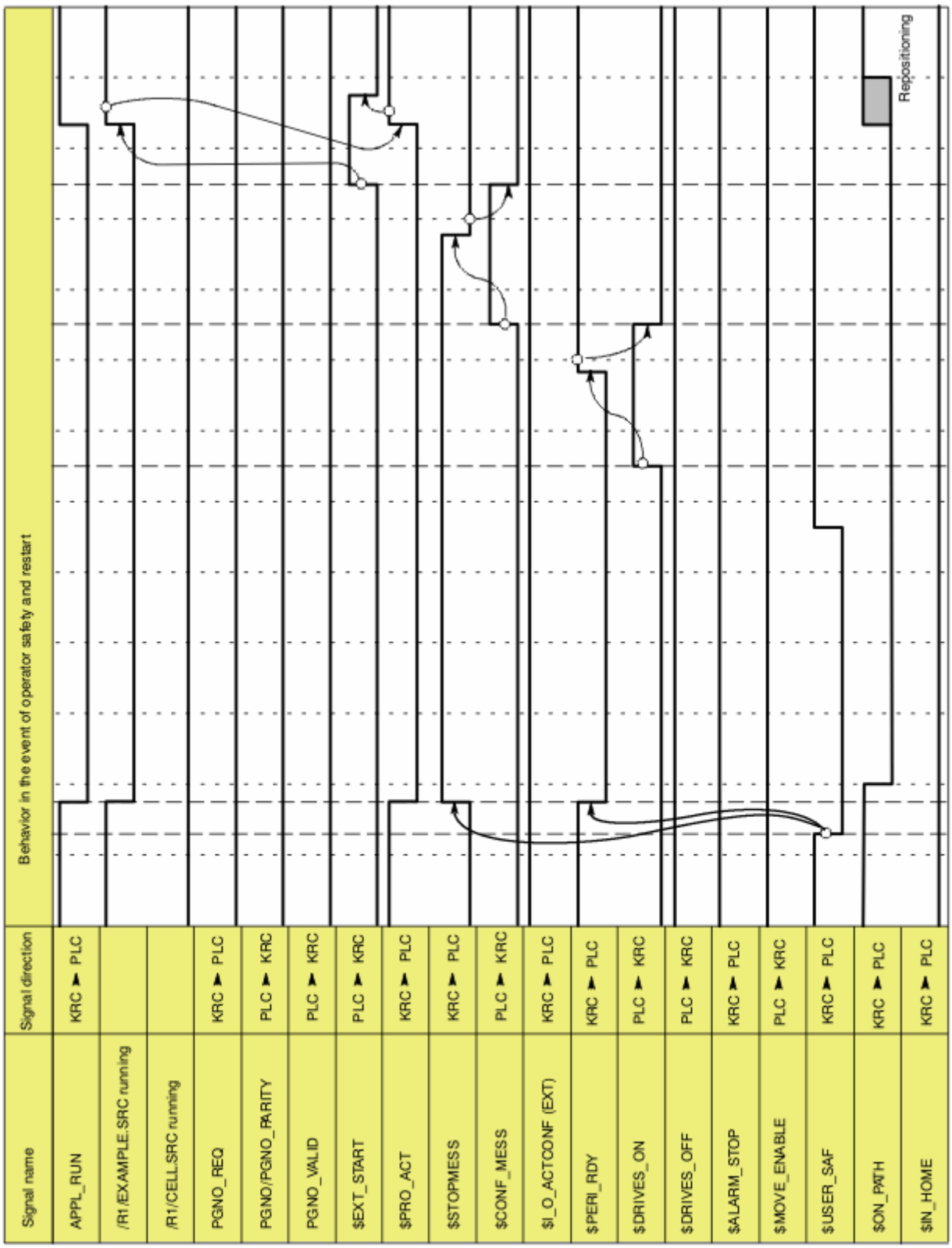


Fig. 6-23: Restart after dynamic braking (operator safety and restart)

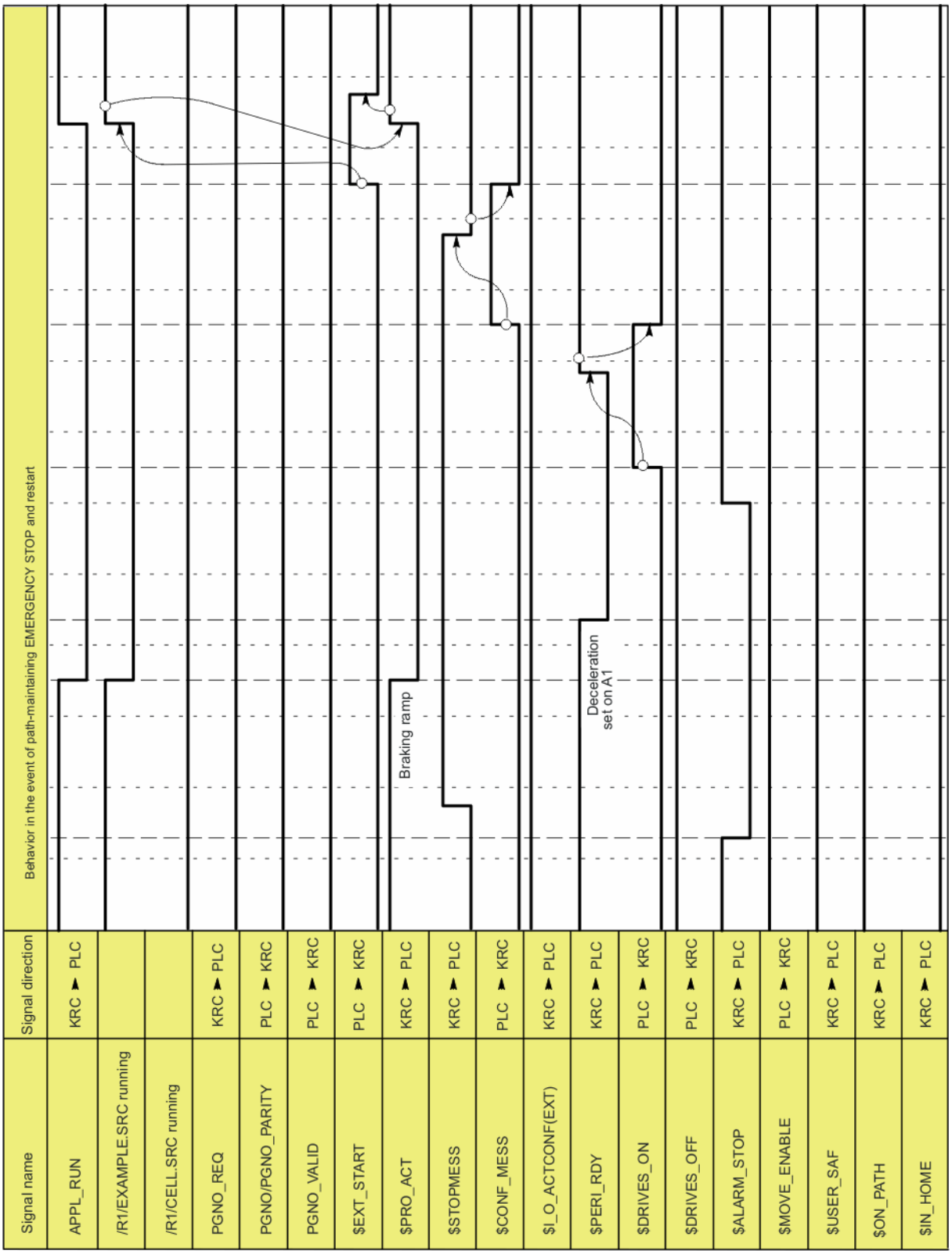


Fig. 6-24: Restart after path-maintaining EMERGENCY STOP

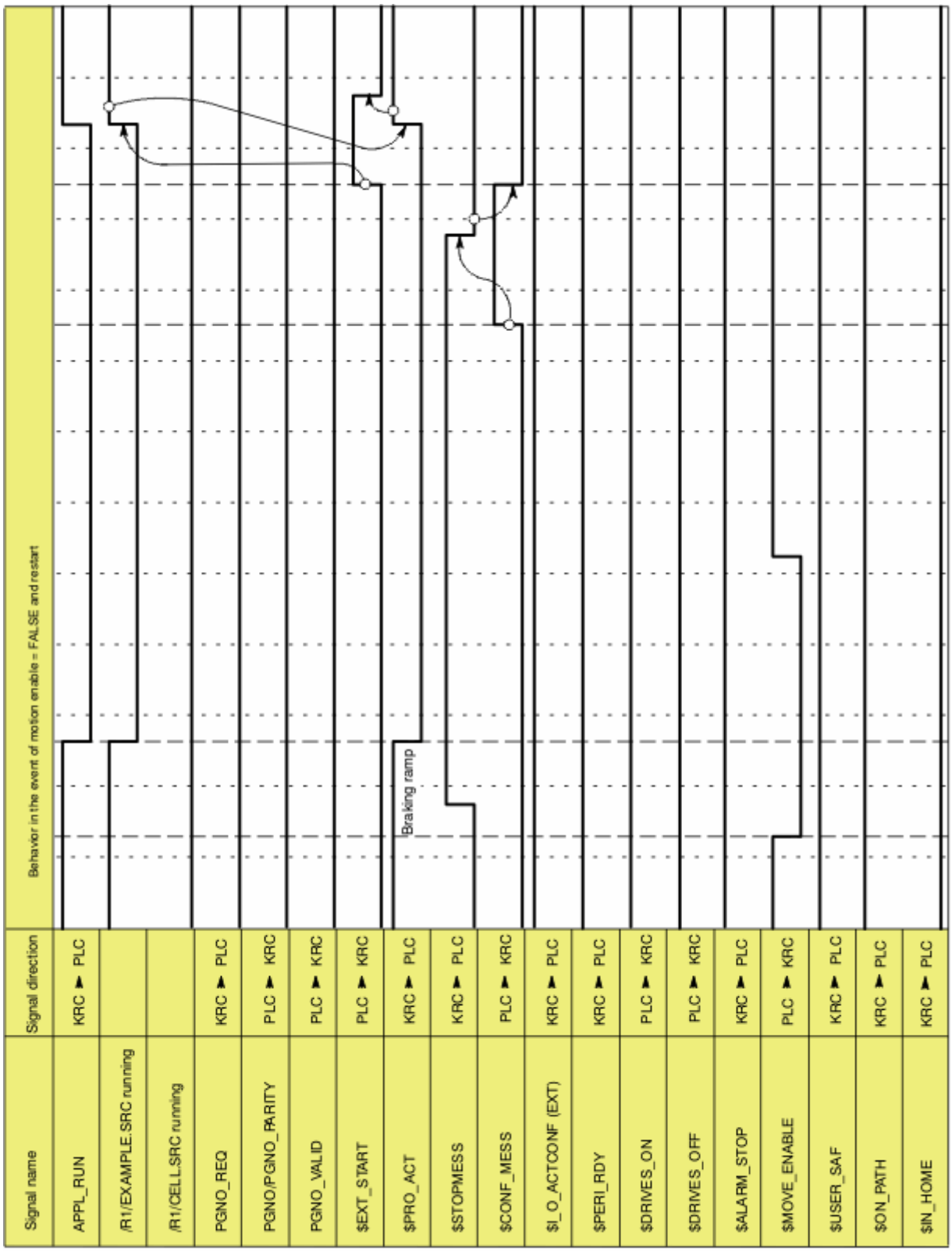


Fig. 6-25: Restart after motion enable

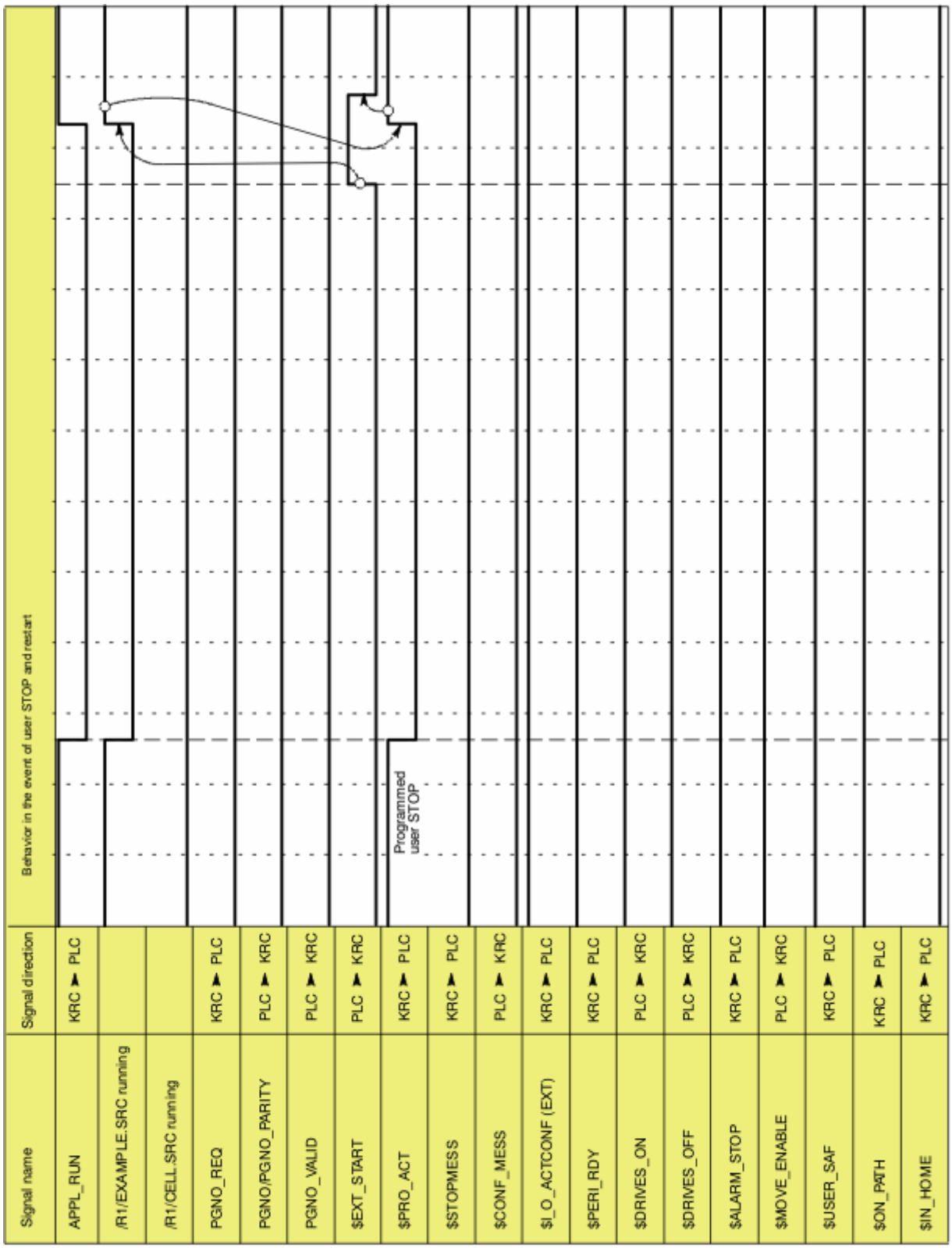


Fig. 6-26: Restart after user STOP

6.20 Torque mode

6.20.1 Overview of torque mode

Description

The “torque mode” function consists of the sub-functionalities “torque limitation” and “deactivation of monitoring functions”.

Torque limitation:

The torques, i.e. the motor current, can be limited for individual axes or multiple axes. Torque limitation enables the following applications:

- The axis can push or pull with a defined torque against a resistance.

  Example:

  Application of a defined pressure on the workpiece by an electric motor-driven spot welding gun.
- The axis can be set to “soft”. It can then be moved by application of an external force. It can be pushed away, for example.

  Examples:

  The robot must grip a workpiece in a press that is then ejected by the press. In order for the robot to be able to yield and absorb the ejector stroke, the affected axis is set to “soft”.

  The robot must set a workpiece down at a point from which it can be pulled into exactly the right position by means of clamps. The robot must be compliant for this.

Deactivation of monitoring functions:

The torque limitation generally results in a relatively large deviation between the command position and the actual position. Certain monitoring functions are triggered by this deviation, although this is undesirable with torque limitation. These regular monitoring functions can thus be deactivated.

Restriction

The following restriction must be taken into consideration if axes are to absorb ejector motions:

A diagonal ejector motion cannot generally be absorbed by switching a single axis to “soft”. Remedy:

- In the case of slightly diagonal ejector motions, a possible remedy is to install the robot with a slight inclination.
- Or contact KUKA Deutschland GmbH.

Inclined installation of the robot is only permissible up to a certain angle of inclination. Further information is contained in the robot operating or assembly instructions.

6.20.1.1 Using torque mode

Torque mode is only possible in program mode, not in manual mode.

WARNING

By default, the robot controller is configured so that only limits that exceed the holding torque of the axis ($HOLDING_TORQUE) can be set. It is nonetheless possible that the axis with limited torque is no longer able to achieve the necessary torque for braking, holding or moving the axis. This can be the case, for example, if the default configuration of the robot controller has been changed or incorrect load data are used. Incorrectly set values can result in unexpected behavior of the robot controller, e.g. motion in a different direction or with different acceleration.
For this reason:

- Only ever limit the torque in small steps, gradually approaching the required limit.
- Do not limit the torque further than necessary.

Failure to take this precaution into consideration may result in death, injuries or damage to property.

If an application requires torque limits that no longer exceed the holding torque of the axis, KUKA Deutschland GmbH must be contacted.

Procedure

1. Set the torque limits for the desired axis and/or deactivate the regular monitoring functions.

   (>>> *6.20.2 "Activating torque mode: SET_TORQUE_LIMITS()" Page 258*)

   If the regular monitoring functions are deactivated, other monitoring functions specially adapted to torque mode are automatically activated.

   (>>> *"Monitoring functions" Page 257*)
2. If the axis is to be set to “soft”: move the axis so that the torque limit becomes active. At the end of the motion, the brakes of this axis remain open.

   Alternatively, a “motion” to the current position can be executed. The robot does not move, but the brakes are released.
3. Optionally: generate a signal indicating that the axis is stationary (e.g. signal to an injection molding machine).
4. Perform the desired action, e.g. move to workpiece and build up pressure or push the axis away.
5. Optionally: wait for a signal to end torque mode again.
6. Deactivate torque mode again.

   (>>> *6.20.3 "Deactivating torque mode: RESET_TORQUE_LIMITS()" Page 261*)

   The torque limits are canceled and the regular monitoring functions are reactivated. Furthermore, the command position is adjusted to the actual position.

Activated/deactivated

Torque mode is considered to be activated in the following case:

- If the upper torque limit is less than or equal to the upper value of the $TORQUE_AXIS_MAX interval.

  (>>> *6.20.5.3 "$TORQUE_AXIS_MAX" Page 264*)
- And/or: If the lower torque limit is greater than or equal to the lower value of the $TORQUE_AXIS_MAX interval.
- And/or: If the regular monitoring functions are deactivated.

Torque mode is considered to be deactivated in the following case:

- If no limits are set or if the limits are invalid. A limit is invalid if it is outside the $TORQUE_AXIS_MAX interval.
- And: If the regular monitoring functions are deactivated.

Automatic deactivation

Torque mode is automatically deactivated in the following cases:

- Program end
- Program reset
- Program deselection
- Block selection (but no deactivation if the target of the block selection is in an interrupt program)
- RESUME (but no deactivation if the RESUME statement returns to an interrupt program)
- Manual mode is activated. Planning is carried out from the current actual position and motion is resumed with full torque.

Monitoring functions

Torque mode generally results in a relatively large deviation between the command position and the actual position. Certain monitoring functions are triggered by this deviation, although this is undesirable in torque mode. These regular monitoring functions can thus be deactivated using SET_TORQUE_LIMITS().

If the regular monitoring functions are deactivated, other monitoring functions specially adapted to torque mode are automatically activated for the actual velocity and following error. If required, user-defined values can be set for these special monitoring functions using SET_TORQUE_LIMITS().

The following messages belong to the regular monitoring functions. They are no longer displayed if the regular monitoring functions are deactivated.

Monitoring	Message no. / message
Following error monitoring	26024: *Ackn. Max. following error exceeded ({Drive}).*
Standstill monitoring	1100: *Stopped {(Axis number)}*
Positioning monitoring	1105: *Positioning monitoring {(Axis number)}*
Monitoring whether motor blocked	26009: *Motor blocked ({Drive}).*

It is also possible, however, to retain the regular monitoring functions in torque mode. This may be useful, for example, if torque mode is used to avoid damage in the case of collisions.

(>>> *6.20.6.2 "Robot program: avoiding damage in the event of collisions" Page 267*)

6.20.1.2 Robot program example: setting A1 to “soft” in both directions

Description

This simple example illustrates the basic principle of torque mode.

In this example, A1 is to be set to “soft” in both directions. For this purpose, both the positive and negative current limits are set to 0 Nm. This allows A1 to be moved by application of an external force.

Program

```
...
1 PTP {A1 10}
2 SET_TORQUE_LIMITS(1, {lower 0, upper 0, monitor #off})
3 PTP {A1 11}
...
4 RESET_TORQUE_LIMITS(1)
5 PTP {A1 -20}
...
```

Line	Description
2	Negative and positive current limits of A1 are set to 0; the regular monitoring functions are deactivated. (The actual velocity and the following error are now monitored with special monitoring functions.)
3	A “motion” is executed to activate torque limitation. (Since both current limits are set to 0, the robot will not actually move.) A1 can now be moved by application of an external force.
4	Deactivate torque mode again for A1. Torque limitation is canceled and the regular monitoring functions are reactivated. Furthermore, the command position is automatically adjusted to the actual position.
5	The robot moves to the next position. (The motion from line 4 is not belatedly executed now, as the command/actual adjustment has been carried out in line 5.)

It is only for A1 that the holding torque is 0 Nm. The limits can therefore not generally be set to 0 for setting an axis to “soft”.

This documentation contains a more detailed example of setting axes to “soft” that can also be applied to other axes.

(>>> *6.20.6.1 "Robot program: setting axis to “soft” in both directions" Page 266*)

6.20.2 Activating torque mode: SET_TORQUE_LIMITS()

Description

This function can be used to perform the following actions for a specific axis:

- Limit the torques in the positive and/or negative direction.
- Deactivate the regular monitoring functions that would be triggered by a higher following error.
- If the regular monitoring functions are deactivated: modify the values for the special monitoring functions.

Function

`SET_TORQUE_LIMITS (`*axis*`: in, `*values*`: in)`

Element	Description
axis	Type: INT Axis to which the statement applies
values	Type: TorqLimitParam Values set for the axis

TorqLimitParam

```
STRUC TorqLimitParam REAL lower, upper, SW_ONOFF moni-
tor, REAL max_vel, max_lag
```

Element	Description
lower	Lower torque limit Unit: Nm (for linear axes: N) Default value: -1E10 (i.e. unlimited)
upper	Upper torque limit Unit: Nm (for linear axes: N) Default value: 1E10 (i.e. unlimited)
monitor	• **#ON** (default): Activates the regular monitoring functions. • **#OFF**: Deactivates the regular monitoring functions. Instead, the monitoring functions max_vel and max_lag are activated.
max_vel	Maximum permissible actual velocity in torque mode (only relevant if the regular monitoring functions are deactivated) Only a positive value may be programmed. Unit: Degrees (for linear axes: mm) Default value (valid for all operating modes): T1 jog velocity * internal safety factor In T1, the maximum velocity with which jogging can be carried out is the default value, even if a higher value is programmed. **Note:** Only set a higher value than the default value if absolutely necessary.
max_lag	Maximum permissible following error in torque mode (only relevant if the regular monitoring functions are deactivated) Only a positive value may be programmed. Unit: Degrees (for linear axes: mm) Default value: 5 degrees (for linear axes: 100 mm) **Note:** Only set a higher value than the default value if absolutely necessary.

lower/upper

When must the upper torque be limited and when must the lower torque be limited?

General: The direction in which the following error is building up must always be limited.

Example: The robot is to be moved up against an obstacle and then stop there. The torque that is thus built up is to be limited.

- If the obstacle appears in the positive direction, `upper` must be set.
- If the obstacle appears in the negative direction, `lower` must be set.

Properties

- SET_TORQUE_LIMITS() can be used in robot programs and in SUB programs running in the system submit interpreter. It cannot be read in SUB programs running in an extended submit interpreter.
- **Advance run stop:** In the robot program, the statement triggers an advance run stop.
- *Values* may remain partially non-initialized. The non-initialized components mean that the existing values are to remain unchanged.
- If both limits are set, `upper` must be >= `lower`.
- If one limit (or both) is already set and the other limit is then set, and if the new limit would result in an empty interval, the new limit value becomes the value for both limits. Example:
  - Already set: `{lower 1, upper 2}`
  - Newly set: `{lower 3}`
  - This results in: `{lower 3, upper 3}`
- It is permissible to set a positive `lower` or a negative `upper` limit.
- The limits set must be greater than the current holding torque $HOLDING_TORQUE. If they are set differently, the robot controller generates an error message that must be acknowledged by the user.

> If an application requires torque limits that no longer exceed the holding torque of the axis, KUKA Deutschland GmbH must be contacted.

- `lower` must be less than or equal to the upper value of the $TORQUE_AXIS_MAX_0 interval.

  `upper` must be greater than or equal to the lower value of the $TORQUE_AXIS_MAX_0 interval.

  If the limits are set differently, the robot controller generates an error message that must be acknowledged by the user.

Examples

Example 1:

For A1, the permissible torque range is limited to the interval 800 … 1,400 Nm.

```
SET_TORQUE_LIMITS(1, {lower 800, upper 1400} )
```

Example 2:

For A3, the upper torque limit is set to 1,200 Nm.

```
SET_TORQUE_LIMITS(3, {upper 1200} )
```

Example 3:

For A1, the regular monitoring functions are (re)activated.

```
SET_TORQUE_LIMITS(1, {monitor #on} )
```

Example 4:

For A1, the permissible torque range is limited to the interval -1,000 … 1,000 Nm. Furthermore, the regular monitoring functions are deactivated and the special monitoring functions are set to user-defined values.

```
SET_TORQUE_LIMITS(1, {lower -1000, upper 1000, monitor #off,
max_vel 10, max_lag 20} )
```

Example 5:

For A1, the permissible torque range is set to -1E10 … 1E10, i.e. the range is unlimited. The regular monitoring functions are (re)activated.

```
SET_TORQUE_LIMITS(1, {lower -1E10, upper 1E10, monitor
#on} )
```

This all corresponds to `RESET_TORQUE_LIMITS(1)`, with the difference that in example 5, the command position is not to adapted to the actual position.

Example 6:

For A1, the lower torque limit is set to a calculated value.

The value has been calculated with the function myCalc() and transferred with the variable myLimits. (In the concrete application, the user must write his own function for this.)

In order for the other components to be non-initialized, the value is pre-initialized with a partially initialized aggregate.

```
DECL TorqLimitParam myParams
...
myParams = {lower 0}
myParams.lower = myCalc()
SET_TORQUE_LIMITS(1, myParams)
```

Example 7:

In this case, the limits are also set to a value that has been calculated with a function. (In the concrete application, the user must write his own function for this.)

The return value of the function is transferred directly, however.

```
DEFFCT TorqLimitParam myCalcLimits()
DECL TorqLimitParam myLimits
...
RETURN myLimits
ENDFCT
...
SET_TORQUE_LIMITS(1, myCalcLimits())
```

6.20.3 Deactivating torque mode: RESET_TORQUE_LIMITS()

Description

This function has the following effect on the selected axis:

- It cancels the limitation of the torques insofar as they were limited.
- It reactivates the regular monitoring functions insofar as they were deactivated.
- It adapts the command position to the actual position.

Function

`RESET_TORQUE_LIMITS (`*axis*`: in)`

Element	Description
axis	Type: INT Axis to which the statement applies

Properties

- RESET_TORQUE_LIMITS() can be used in robot programs and in SUB programs running in the system submit interpreter. It cannot be read in SUB programs running in an extended submit interpreter.
- **Advance run stop:** In the robot program, the statement triggers an advance run stop. This cannot be masked with CONTINUE!

Alternative

If no command/actual value adjustment is required, torque mode can also be deactivated with SET_TORQUE_LIMITS instead of RESET_TORQUE_LIMITS:

```
SET_TORQUE_LIMITS(1, {lower -1E10, upper 1E10, monitor
#on} )
```

- Advantage: Can be used during a motion (“on the fly”).
- Disadvantage: If the torque limitation has resulted in a relatively large following error, the robot accelerates very fast. This can trigger monitoring functions and stop the program.

Deactivation by means of SET_TORQUE_LIMITS is not suitable in most cases.

6.20.4 Interpreter specifics

Description

- SET_TORQUE_LIMITS() and RESET_TORQUE_LIMITS() can be used in robot programs and in submit programs.
- The statements are interpreter-specific, i.e. they only work in the interpreter in which they have been used.
- SET_TORQUE_LIMITS() first takes effect when the axis is moved for the interpreter that generates the statement. Example:
  1. Torque mode is activated in the robot program for an external axis.
  2. The external axis is moved by a submit program. Torque mode has no effect.
  3. The external axis is moved by a robot program. Torque mode takes effect.
- If torque mode is already active, SET_TORQUE_LIMITS() takes effect immediately.
- SET_TORQUE_LIMITS() works immediately if a motion is active. For this reason, the torque limits can be set at any time in robot programs, both inside and outside an interrupt, and the monitoring functions can be activated and deactivated.

  It is also possible to use torque mode inside an interrupt program only. (If RESET_TORQUE_LIMITS() is used, it may subsequently be necessary to return to the interrupt position with PTP $AXIS_RET.)

  (>>> *6.20.6.3 "Robot program: torque mode in the interrupt" Page 268*)

- When a torque-driven axis “changes owner”, the command position is adjusted to the actual position.

  “Change of owner” means: an interpreter has moved the axis in torque mode (and thus “owns” it). While torque mode is active, the axis is moved by a different interpreter.

  The main application here is: jogging after a program has been interrupted in torque mode.

Example

The following example shows when SET_TORQUE_LIMITS() is effective, depending on whether torque mode is already active or not.

Initial situation (default): the monitoring functions are activated.

```
1 SET_TORQUE_LIMITS(1, {monitor #off})
2 HALT
3 PTP_REL {A1 10}
4 HALT
5 SET_TORQUE_LIMITS(1, {monitor #on})
6 HALT
7 PTP_REL {A1 15}
```

Line	Description
1	The monitoring functions for A1 are deactivated.
2	Here the monitoring functions are still activated.
3	The axis is moved. From here on, the statement SET_TORQUE_LIMITS is effective.
4	The monitoring functions are deactivated.
5	The monitoring functions are activated.
6	Here the monitoring functions are already activated. Because torque limitation was already active, the statement took effect immediately and not just after the next motion of this axis.

6.20.5 Diagnostic variables for torque mode

All these variables and constants are write-protected.

Their value is not dependent on the interpreter.

6.20.5.1 $TORQUE_AXIS_ACT

Variable

$TORQUE_AXIS_ACT[*axis number*]

Data type: REAL

Description

Current motor torque for axis [*axis number*]

Unit: Nm (for linear axes: N)

The value is only relevant if the brakes are released. If the brakes are applied, it is virtually zero.

Advance run stop: In the robot program, the variable triggers an advance run stop.

(>>> *6.20.5.6 "Comparison: $TORQUE_AXIS_ACT and $HOLDING_TORQUE " Page 266*)

$BRAKE_SIG

The state of the brakes can be displayed by means of the system variable $BRAKE_SIG. The value of $BRAKE_SIG is a bit array: bit 0 corresponds to A1, bit 6 corresponds to E1.

- Bit n = 0: Brake is closed.
- Bit n = 1: Brake is open.

6.20.5.2 $TORQUE_AXIS_MAX_0

Constant

$TORQUE_AXIS_MAX_0[*axis number*]
Data type: REAL

Description

Maximum permanent motor torque for axis [*axis number*] at velocity 0
The value specifies an interval: from -*value* to +*value*.
Unit: Nm (for linear axes: N)
Advance run stop: does not trigger an advance run stop.

`lower` must be less than or equal to the upper value of the $TORQUE_AXIS_MAX_0 interval.
`upper` must be greater than or equal to the lower value of the $TORQUE_AXIS_MAX_0 interval.
If the limits are set differently, the robot controller generates an error message that must be acknowledged by the user.

6.20.5.3 $TORQUE_AXIS_MAX

Constant

$TORQUE_AXIS_MAX[*axis number*]
Data type: REAL

Description

Absolute maximum motor torque for axis [*axis number*]
The value specifies an interval: from -*value* to +*value*.
Unit: Nm (for linear axes: N)
Advance run stop: does not trigger an advance run stop.

6.20.5.4 $TORQUE_AXIS_LIMITS

Variable

$TORQUE_AXIS_LIMITS[*axis number*]
Data type: TorqLimitParam

Description

Currently active motor torque limitation for axis [*axis number*]
Unit: Nm (for linear axes: N)

The variable is primarily intended for diagnosis via the variable correction function or variable overview.

Characteristics:

- If there are currently no limits active, `upper` and `lower` remain non-initialized.
- The component `monitor` is always initialized unless the axis does not exist.

  This is relevant, for example, in the case of 4-axis and 5-axis robots: if the entire array is displayed, the non-existent axes can be easily identified.

  Non-existent external axes are simply not displayed when the entire array is displayed.
- `Max_vel` and `max_lag` are non-initialized if `monitor` = #ON, as the regular monitoring functions are active in this case.

  If `monitor` = #OFF, the values of `max_vel` and `max_lag` are displayed. The display is irrespective of whether they have been set explicitly in the current program or whether the default values are being used.

> The fact that certain components remain non-initialized under certain conditions, simplifies diagnosis for the user.
> If the variable $TORQUE_AXIS_LIMITS[] is accessed via KRL, however, the robot controller may regard the access as “invalid”. Recommendation: Check the state of the variable with VARSTATE() prior to access.

Advance run stop: In the robot program, the variable triggers an advance run stop.

6.20.5.5 $HOLDING_TORQUE

Variable

$HOLDING_TORQUE[*axis number*]

Data type: REAL

Description

Holding torque for the robot axis [*axis number*]

Unit: Nm

The holding torque refers to the current actual position of the axis and the current load.

(For external axes, the value 0 N is always returned.)

Advance run stop: In the robot program, the variable triggers an advance run stop.

(>>> *6.20.5.6 "Comparison: $TORQUE_AXIS_ACT and $HOLDING_TORQUE " Page 266*)

> If the upper and lower torque limits are set to $HOLDING_TORQUE for all axes, the robot must remain stationary when the brakes are released.
> If this is not the case, i.e. if the robot drifts, the load is not correctly configured.

6.20.5.6 Comparison: $TORQUE_AXIS_ACT and $HOLDING_TORQUE

In the case of a robot that is stationary with the brakes released, $TORQUE_AXIS_ACT is not equal to $HOLDING_TORQUE, although this might be assumed.

Characteristics of **$HOLDING_TORQUE**:

- A calculated value that does not take friction into consideration. Control effects have no influence on the value.
- Is only dependent on the current position of the axis. Thus remains unchanged if the position remains constant.

Characteristics of **$TORQUE_AXIS_ACT**:

- Calculated from the actual currents. Friction and control effects thus affect the value.
- Can change if the position remains constant.

The discrepancy between $HOLDING_TORQUE and $TORQUE_AXIS_ACT is less than or equal to the current friction if the robot remains stationary for at least 1 second. A precondition is that the load data are correct.

6.20.6 Other examples

6.20.6.1 Robot program: setting axis to “soft” in both directions

Description

The robot must grip a workpiece in a press that is then ejected by the press. In order for the robot to be able to yield and absorb the ejector stroke, the axis is set to “soft”.

For this, the torque limits must be set to a very small interval around the holding torque. (It is only for A1 that the holding torque = 0 Nm. The limits can therefore not generally be set to 0 for setting an axis to “soft”.)

Assumption for this example: a rotation about the axis [*ideal_axis*] moves the gripper almost exactly in the ejector direction.

Program

```
DECL TorqLimitParam myLimits
DECL INT ideal_axis
...
myLimits.monitor = #off
myLimits.lower = $holding_torque[ideal_axis] - 10
myLimits.upper = $holding_torque[ideal_axis] + 10
SET_TORQUE_LIMITS(ideal_axis, myLimits)
PTP $AXIS_ACT
OUT_SIGNAL_SOFT = TRUE
WAIT FOR IN_SIGNAL_EJECTED
RESET_TORQUE_LIMITS(ideal_axis)
OUT_SIGNAL_SOFT = FALSE
WAIT FOR IN_SIGNAL_NEXTMOVE
...
```

Line	Description
3 …6	For the axis [*ideal_axis*], the regular monitoring functions are deactivated and the torques are limited to a very small interval around the holding torque.
7	Execute a “motion” to the current position to activate reduction of the current limits. The axis [ideal_axis] can now be moved by the ejector of the press.
8	Signal to the press controller that the axis is ready for the ejection.
9	Wait for signal from press controller that the workpiece has been ejected.
10	Cancels torque limitation and reactivates the regular monitoring functions. Furthermore, adjusts the command position to the actual position.
11	Signal to the press controller that the axis is no longer ready for an ejection.
12	If required: Wait for a signal that motion away from the position is allowed.

6.20.6.2 Robot program: avoiding damage in the event of collisions

Description

Torque limitation can be used to avoid damage in the event of collisions.

- Advantage: It is assured that the robot only presses against the obstacle with a defined, limited force.
- Disadvantage: The robot becomes sluggish. High accelerations are no longer possible.

Program

The robot fetches workpieces from a box. During the motion to points P7, P8 and P9, the possibility cannot be ruled out of the robot with the workpiece colliding with the box. It must be ensured that the robot does not press so hard that damage can result. For this, the forces are limited before the critical points.

The regular monitoring functions are deactivated. This is not because they would otherwise be triggered unnecessarily; on the contrary, they are not strict enough for this example. Instead, one of the special monitoring functions is set to a very low value. (Depending on the specific application, it may also be useful to use the regular monitoring functions.)

```
...
1 DECL TorqLimitParam myParams
...
2 FOR i = 1 to 6
3 myParams.lower = $holding_torque[i] - 500
4 myParams.upper = $holding_torque[i] + 500
5 myParams.monitor = #off
6 myParams.max_lag = 0.1
7 SET_TORQUE_LIMITS(i, myParams)
8 ENDFOR
9 $acc.cp = my_low_acceleration
10 $vel.cp = my_low_velocity
11 LIN P7
```

```
12 LIN P8
13 LIN P9
14 FOR i = 1 to 6
15 myParams.lower = -1E10
16 myParams.upper = 1E10
17 myParams.monitor = #on
18 SET_TORQUE_LIMITS(i, myParams)
19 ENDFOR
20 $acc.cp = my_high_acceleration
21 $vel.cp = my_high_velocity
22 LIN P10
...
```

Line	Description
2 … 7	The torques for A1 ... A6 are limited.
3, 4	The limits are set to a relatively small interval centered on the holding torque.
5, 6	The regular monitoring functions are deactivated. `Max_lag = 0.1` has the effect of triggering a stop in the case of a following error as low as 0.1°.
9, 10	Acceleration and velocity are reduced so that the robot approaches the critical point slowly.
11 …13	Points at which a collision could occur. If a collision occurs, the monitoring function `max_lag` is triggered and the system operator can intervene.
After the critical section:	
14 …19	Torque limitation is deactivated. SET_TORQUE_LIMITS can be used here: the robot only reaches this point if it has passed the critical points without collision. In this case, no following error has built up and command/actual value adjustment is not required.
20, 21	Acceleration and velocity are reset to the previous higher values.
22	Uncritical point

6.20.6.3 Robot program: torque mode in the interrupt

Description

This demonstrates that torque mode can be used completely inside an interrupt program. The example is not primarily a practical one, but is intended to show that this use is generally possible.

Program

The robot is to determine the position of a workpiece whose possible location is known, but not its exact position. First, a proximity sensor signals to the robot controller when the robot is in the vicinity of the workpiece. On the basis of this sensor signal, an interrupt program is then called.

The actual search is carried out in the interrupt program: A1 moves towards the workpiece. Torque mode is activated for A1 beforehand. The robot comes to a standstill where it makes contact with the workpiece due to the reduced torques: the workpiece has been “found”. This position can now be saved as the position of the workpiece.

```
...
1 LIN P1
2 INTERRUPT DECL 1 WHEN sensor_signal==true DO search()
3 INTERRUPT ON 1
4 LIN P2
5 INTERRUPT OFF 1
...
-------------------------
6 DEF search()
7 ...
8 BRAKE
9 SET_TORQUE_LIMITS(1,{lower 1000, upper 1000, monitor
#off})
10 PTP_REL {A1 10}
11 RESET_TORQ_LIMITS(1)
12 piece_found = $POS_ACT_MES
13 PTP $AXIS_RET
14 END
```

Line	Description
1 … 4	On the way to P2, the sensor is to signal when the robot is in the vicinity of the workpiece.
2	When the sensor signal is received, the subprogram search() is called.
In the subprogram:	
8	The current motion is stopped as soon as search() is executed.
9	Sets torque limits for A1, deactivates regular monitoring functions.
10	A1 moves. The robot comes to a standstill where it makes contact with the workpiece due to the reduced torques. Once the robot has reached its command position {A1 10}, the next program line is executed. (The fact that the actual position deviates from the command position has no effect, as the regular monitoring functions have been deactivated.)
11	Cancels torque limitation and reactivates the regular monitoring functions. Furthermore, adjusts the command position to the actual position.
12	The current position of the robot now indicates the position of the workpiece. This is saved here in a variable.
13	Returns to the position at which the robot left the path in the main program.

6.20.6.4 Robot program: servo gun builds up pressure

Description

This program shows how torque mode can be activated by means of a trigger. (Comparable programs are used in the background in the KUKA.ServoGun technology packages. They thus do not need to be programmed by the user.)

Program

Main program:

```
1 DEF SPOT()
2 DECL BOOL error_occurred
...
3 Interrupt DECL 1 WHEN $stopmess DO resume_subprog()
4 Interrupt ON 1
5 REPEAT
6 error_occurred = false
7 SPOT_MOVE()
8 UNTIL error_occurred == false
...
```

Line	Description
3	If an error occurs, resume_subprog() is to be called.
7	The weld program SPOT_MOVE() is called.
5 … 8	If an error has occurred (i.e. if error_occurred == true), SPOT_MOVE() is repeated.

Weld program:

```
1 DEF SPOT_MOVE()
...
2 TorqLimWeld = {lower -1000, upper 1000 , monitor #off}
3 i = 6+EG_EXTAX_ACTIVE
...
4 LIN P_APPROX C_DIS
5 $VEL_EXTAX[EG_EXTAX_ACTIVE]=EG_MAX_CONST_VEL[EG_EXTAX_AC-
TIVE]
6 LIN P_APPROX C_DIS
7 TRIGGER WHEN DISTANCE=0 DELAY=50 DO SET_TORQUE_LIMITS(i,
TorqLimWeld) PRIO = -1
8 LIN P_PART C_DIS
9 TRIGGER WHEN DISTANCE=0 DELAY=50 DO START_TIMER_SPOT()
PRIO=82
10 LIN P_PRESSURE C_DIS
11 LIN P_WELD
12 WAIT FOR EG_TRIGGER_END
13 RESET_TORQUE_LIMITS(i)
14 Interrupt OFF 1
15 LIN P_PART C_DIS
16 END
```

Line	Description
7	Shortly before the gun touches the workpiece, the torques are reduced.
9 … 11	Build up pressure and then weld.
12	The weld timer signals the end of welding.
13	Cancels torque limitation and reactivates the regular monitoring functions. Furthermore, adjusts the command position the actual position.

Interrupt program in the event of an error:

```
1 DEF resume_subprog()
2 BRAKE
3 Suppress_repositioning()
4 HALT
5 error_occurred = true
6 RESUME
7 END
```

Line	Description
3	Normally, in the case of a restart after HALT, the robot is repositioned to the position at which the interrupt was triggered. (This is because of $STOPMESS.) Suppress_repositioning() prevents this repositioning. Suppress_repositioning() may be useful, depending on the application, but not necessarily.
5	Set error_occurred to TRUE so that the REPEAT loop in the main program is repeated.
6	RESUME deactivates torque mode. Jump back to line 8 in the main program.

6.20.6.5 Submit program: servo gun builds up pressure

Precondition

- E1 is already asynchronous with `ASYPTP {E1 10}`.
- or: $ASYNC_MODE is configured in such a way (bit 0 = 1) that the axis is implicitly set to synchronous mode in the case of ASYPTP in the submit program.

Program

```
...
1 IF $PRO_STATE1==#P_FREE
2 SET_TORQUE_LIMITS(7,{upper 1000, monitor #off })
3 ASYPTP {E1 10}
...
4 RESET_TORQUE_LIMITS(7)
5 ASYPTP {E1 -10}
6 ENDIF
...
```

Line	Description
1	Ensure that no robot program is selected.
2	Limit the positive torque and deactivate the regular monitoring functions.
3	Motion towards end point {E1 10} behind the workpiece. The pressure on the workpiece builds up.
4	Cancels torque limitation and reactivates the regular monitoring functions. Furthermore, adjusts the command position to the actual position. The interpreter waits in RESET_TORQUE_LIMITS(7) until the asynchronous motion has been completed. Only then does it perform the command/actual value adjustment. It is therefore not necessary to program `WAIT FOR $ASYNC_STATE == #IDLE` before “RESET…”.
5	Reopen the gun.

6.21 Event planner: configuring a data comparison

Precondition

- User rights: Function group **General configuration**

Procedure

1. In the main menu, select **Configuration** > **Miscellaneous** > **Event planner**.
2. Expand the tree structure in the left-hand section of the window and select the desired action.

   Scheduled actions

   - **T1 and T2 consistency**: Compare data in operating modes T1 and T2 at regular intervals.
   - **AUT and EXT consistency**: Compare data in the automatic operating modes at regular intervals.

   Conditional actions

   - **Logic consistency**: Compare data after a change of operating mode or after online optimizing.

     Online optimizing is the modification of the parameters of a program during operation.
3. Make the desired settings in the right-hand part of the window. Depending on the selected action, various parameters are available for selection, e.g. various time intervals.
4. Press **Save**.

6.22 Brake test

6.22.1 Overview of the brake test

Description

Each robot axis has at least one holding brake integrated into the motor. The brake test checks to see if the braking torque is sufficiently high, i.e. whether it exceeds a certain minimum value. The minimum value for the individual motor types is stored in the machine data and cannot be configured. (The brake test does not calculate the absolute value of the braking torque.)

It is advisable to carry out the brake test when the robot is at operating temperature. This is the case after approx. 1 h in normal operation.

Activation + configuration

- The “brake test” functionality is automatically active if a safety option is installed and safe monitoring is activated.
- If the brake test is not automatically active, the user has the option of manually activating it (in WorkVisual or on the robot controller).
- The axes to be checked in the brake test can be configured (in WorkVisual or on the robot controller).
- The cycle time can be configured (in WorkVisual or on the robot controller).

If the brake test is not automatically active, the operator must carry out a hazard assessment to determine whether it is necessary to activate the brake test for the specific application.

If the brake test is active, the operator must perform a hazard assessment to determine the following:

- Which axes need to be tested
- What cycle time needs to be defined

It is irrelevant whether the brake test is automatically active or it is activated manually. The hazard assessment is required in both cases.

Request

Events which request the execution of a brake test

If the brake test is active, the following events request the execution of a brake test:

- Input $BRAKETEST_REQ_EX is set externally, e.g. by a PLC (external request)
- Robot controller boots with a cold start (internal request)
- Brake test cycle time has elapsed (internal request)

  The default cycle time is 46 h. It elapses when the drives have been in servo-control for a total of 46 h.
- Status message from the automatic brake check:

  Brake defective, {Axis} permanently under servo control

Response following a request

1. If a request is present, the robot controller generates the following message: *Brake test required*.

   The robot can be moved for another 2 hours. (This is referred to as the monitoring time)
2. The brake test must be performed within the monitoring time. Once the brake test has been performed successfully, the cycle time restarts.
3. If the brake test is not performed, the robot stops once the monitoring time expires. The robot controller generates the following acknowledgement message: *Cyclical check for brake test request not made*.

   The message cannot be acknowledged externally (by the PLC), but must be acknowledged on the smartPAD.

   Once the message has been acknowledged, the robot can be moved for another 2 hours.

At the time of the brake test, a simulation can be switched on, for example via $SIMULATED_AXIS, $SIMULATED_COOP_ROBOTS or $SERVO_SIM. The simulated axes are not included in the brake test. Simulated axes must be removed from the simulation and tested before the end of the cycle time. Otherwise, the robot stops and the robot controller generates the following acknowledgement message: *Cyclical check for brake test request not made*. Once the message has been acknowledged, the robot can be moved for another 2 hours.

Active and requested axes

“Active axes” are those axes selected in the **Active Configuration** column in the **Brake test configuration** window.

“Requested axes” are the active axes for which there is currently a brake test request.

6.22.2 Sequence when testing a brake

The brake test checks the brakes to be tested one after the other.

1. From the start position of the brake test, the axis to be tested moves in the direction in which the software limit switch is situated further away, and then moves back. The gravitation and friction of the axis to be tested are determined during this motion.

   Rotational axes move a maximum of 5° in the direction of the software limit switch; linear axes a maximum of 10 cm.
2. When the axis has returned to its start position, the brake closes and the motor torque exerted against the closed brake is increased.

   The results of the brake test are shown in the message window.
3. If a brake has been identified as being defective, the robot moves to the parking position following confirmation.

   If a brake has reached the wear limit, the robot controller indicates this by means of a message. A worn brake will soon be identified as defective. Until then, the robot can be moved without restrictions.

If an axis is equipped with additional brakes, the main brake is tested first.

6.22.3 Programs for the brake test

The programs are located in the directory C:\KRC\ROBOTER\KRC\R1\TP\BrakeTest.

Program	Description
BrakeTestReq.src	Performing the brake test cyclically (via program): • All requested axes can be tested in one cycle using the program. For this purpose, the program is called without parameters. • A selection of the requested axes can also be tested using the program. The desired axes are transferred as parameters when calling the program. This enables the brake test to be divided into multiple shorter cycles. **Note:** This allows, for example, small breaks in the application to be utilized for testing individual axes. (>>> *6.22.5.1 "Performing a brake test for requested axes (cyclically via program)" Page 283*)
	BrakeTestReq.src can also be selected manually. All active axes are tested. (>>> *6.22.5.2 "Performing a brake test for active axes (manually)" Page 285*)
BrakeTestAxes.src	With the program, axes for which there is no brake test request can be tested. In particular, it also enables the testing of axes which cannot be activated for the brake test and thus cannot be tested via BrakeTestReq.src. Couplable axes fall into this category, for example. (>>> *6.22.5.3 "Performing a brake test for further axes (e.g. couplable axes)" Page 286*)
BrakeTestPark.src	The parking position of the robot must be taught in this program. If, during the brake test, a brake has been identified as being defective, the robot is moved to the parking position following confirmation.

Program	Description
BrakeTestStart.src	The start position of the brake test can be taught in this program. The robot starts the brake test from this position. If the start position is not taught, the robot performs the brake test at the actual position.
BrakeTestBack.src	The end position of the brake test can be taught in this program. The robot moves to this position after the brake test. If the end position is not taught, the robot remains at the actual position after the brake test.

6.22.4 Overview of the brake test setup

Step	Description
In WorkVisual or on the robot controller:	
1	Activate the brake test; define the cycle time and axes • On the robot controller: (>>> *6.22.4.1 "Activating the brake test, defining the cycle time and axes" Page 275*) • In WorkVisual: Information about activating the brake test in WorkVisual is contained in the WorkVisual documentation.
On the robot controller:	
2	Configure input and output signals for the brake test. (>>> *6.22.4.3 "Configuring input and output signals for the brake test" Page 277*)
3	Teach positions for the brake test. (>>> *6.22.4.5 "Teaching positions for the brake test" Page 281*)
4	Test the sequence. (>>> *6.22.4.6 "Testing the sequence in the case of defective brakes" Page 282*)

6.22.4.1 Activating the brake test, defining the cycle time and axes

Precondition

- User group “Safety maintenance” or higher
- T1 or T2 mode
- No program is selected.

Procedure

1. In the main menu, select **Configuration** > **Brake test configuration**.
   The **Brake test configuration** window opens.
   (>>> *6.22.4.2 "“Brake test configuration” window" Page 276*)
2. If necessary, activate or deactivate the check box for **Forced**in the **Current configuration** column.
3. Also make the desired settings for the cycle time and axes in the **Current configuration** column.
4. Press the **Activate** button.

The message *Reconfiguration in progress ...* is displayed. The message disappears automatically when reconfiguration has been completed. The new settings for the brake test are now saved and valid.

6.22.4.2 “Brake test configuration” window

Fig. 6-27: “Brake test configuration” window

Element	Description
Configurations are identical	• LED lights up green: The settings in the **Active Configuration** and **Current configuration** columns are identical. • LED lights up red: The settings are not identical.
Checksum	Checksum of the brake test configuration in the corresponding column • Checksums in both columns are identical: The settings in the columns are identical. Corresponds to the green LED for **Configurations are identical**. • Checksums not identical: The settings are not identical. Corresponds to the red LED.

Element	Description
Current configuration	The settings can be modified in this column. The most recent modifications are shown.
Active Configuration	This column displays the valid settings. A check mark next to an axis means that the axis is selected for the brake test. This column is used for display purposes only. Modifications are not possible here.
Forced	• Check box not active: Automatic response of the brake test – A safety option is installed and safe monitoring is active. The brake test is thus automatically active. – Or: No safety option is installed or safe monitoring is not active. The brake test is thus automatically inactive. • Check box active: Forced response The brake test has been explicitly activated by the user (Here or in WorkVisual). It is active, irrespective of safety options or safe monitoring. **Note:** The **Forced** check box WITHOUT a check mark does not indicate whether the brake test is active or not! The state of the brake test is indicated in the **Safety configuration** window on the **Common** tab: • **Activated**: The brake test is active for at least one axis. • **Deactivated**: The brake test is not active for any axis.
Cycle time [h]	The cycle time specifies the interval at which the brake test is to be executed. • 1 … 1000 Default: 46. Unit: hours
[*Axis no.*]:[*Robot type*]	The robot axes and external axes for which the brake test is to be executed can be selected here. By default, all axes are selected. The following external axes cannot be selected: 1. External axes that are simulated 2. External axes that are configured as couplable 3. External axes with motors that are grouped together into a coupling group
Activate	Saves the settings of the **Current configuration** column. The system then automatically performs a reconfiguration. Once reconfiguration is completed, new settings are valid and are displayed in the **Active Configuration** column. **Activate** is only available if the 2 columns have different settings.
Reset to active configuration	Resets the **Current configuration** column to the settings from the **Active Configuration** column. Only available if the 2 columns have different settings.

6.22.4.3 Configuring input and output signals for the brake test

Description

All signals for the brake test are declared in the file $machine.dat in the directory KRC:\STEU\MADA.

WARNING

These signals are not redundant in design and can supply incorrect information. Do not use these signals for safety-relevant applications.

Precondition

- User rights: Function group **Critical KRL program changes**

Procedure

1. Open the file $machine.dat in the directory KRC:\STEU\MADA in the Navigator.
2. Assign inputs and outputs.
3. Save and close the file.

$machine.dat

Extract from the file $machine.dat (with default settings, without comments):

```
...
SIGNAL $BRAKETEST_REQ_EX $IN[1026]
SIGNAL $BRAKETEST_MONTIME FALSE
...
SIGNAL $BRAKETEST_REQ_INT FALSE
SIGNAL $BRAKETEST_WORK FALSE
SIGNAL $BRAKES_OK FALSE
SIGNAL $BRAKETEST_WARN FALSE
...
```

Signals

There is 1 input signal. By default, it is routed to $IN[1026].

The output signals are preset to FALSE. There is no need to assign output numbers to them.

Signal	Description
$BRAKETEST_REQ_EX	Input • **TRUE** = brake test is being requested externally (e.g. by PLC). The robot controller confirms the signal with $BRAKETEST_REQ_INT = TRUE and generates message 27004. • **FALSE** = brake test is not being requested externally.
$BRAKETEST_MONTIME	Output • **TRUE** = robot was stopped due to elapsed monitoring time. Acknowledgement message 27002 is generated. • **FALSE** = acknowledgement message 27002 is not active. (Not generated, or has been acknowledged.)
$BRAKETEST_REQ_INT	Output • **TRUE** = message 27004 is active. The signal is not set to FALSE again until a brake test is carried out with a positive result, i.e. with message 27012. • **FALSE** = brake test is not requested (either internally or externally).

Signal	Description
$BRAKETEST_WORK	Output • **TRUE** = brake test is currently being performed. • **FALSE** = brake test is not being performed. If no defective brakes have been detected, message 27012 is generated. Edge **TRUE** → **FALSE**: • Test was successfully completed. No brake is defective. Message 27012 is generated. • Or at least 1 defective brake was detected and the robot has moved to the parking position. • Or the program was canceled during execution of the brake test.
$BRAKES_OK	Output • Edge **FALSE** → **TRUE**: Output was set to FALSE by the previous brake test. The brake test was carried out again and no defective brake was detected. • Edge **TRUE** → **FALSE**: A brake has just been detected as defective. Message 27007 is generated. This signal does not take any couplable axes into account.
$BRAKETEST_WARN	Output • Edge **FALSE** → **TRUE**: At least 1 brake has been detected as having reached the wear limit. Message 27001 is generated at the same time. • Edge **TRUE** → **FALSE**: Output was set to TRUE by the previous brake test. The brake test was carried out again and no worn brake was detected. This signal does not take any couplable axes into account.

Messages

No.	Message
27000	*Test of brakes {Axis bit mask} not executed because axes are simulated*
27001	*Brake {Brake no.}{Axis no.} has reached the wear limit*
27002	*Cyclical check for brake test request not made*
27003	*Brake test for axes {Axis bit mask} required*
27004	*Brake test required*
27007	*Insufficient holding torque of brake {Brake no.}{Axis no.}*
27009	*Brake {Brake no.}{Axis no.} OK*
27010	*Unable to verify performance of brake {Brake}{Axis}*
27012	*Brake test successful*

6.22.4.4 Signal diagram of the brake test – examples

Example 1

The signal diagram for the brake test applies in the following case:

- No brake has reached the wear limit.
- No brake is defective.

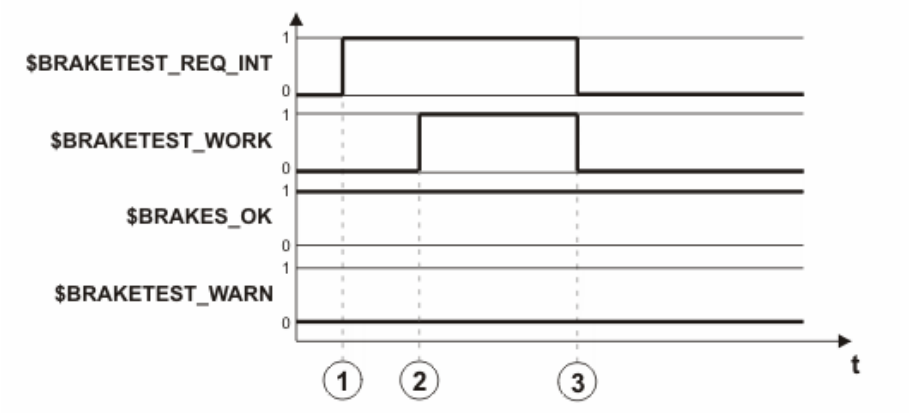


Fig. 6-28: Signal diagram: brakes OK

Item	Description
1	The brake test is requested.
2	Automatic call of the program BrakeTestReq.src Start of the brake test
3	The brake test is completed.

Example 2

The signal diagram for the brake test applies in the following case:

- Brake A2 is worn.
- Brake A4 is defective.

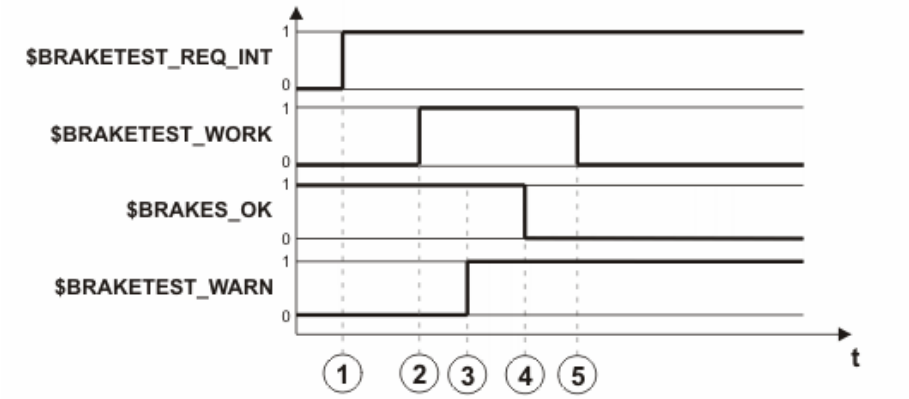


Fig. 6-29: Signal diagram: brakes not OK

Item	Description
1	The brake test is requested. $BRAKETEST_REQ_INT is not set to FALSE again until a brake test is carried out with a positive result.
2	Automatic call of the program BrakeTestReq.src Start of the brake test
3	Brake A2 is tested: brake is worn.
4	Brake A4 is tested: brake is defective.

Item	Description
5	The robot has been moved to the parking position or the program has been canceled.

6.22.4.5 Teaching positions for the brake test

Description

Start position and end position

The start position and end position can be taught.

- If the start position is not taught, the robot performs the brake test at the actual position.
- If the end position is not taught, the robot remains at the actual position after the brake test.

Parking position

The parking position must be taught.

If, during the brake test, a brake has been identified as being defective, the robot is moved to the parking position following confirmation.

WARNING

If a brake is identified as being defective and the drives are deactivated, the robot may sag. For this reason, no stop may be triggered during the motion to the parking position. The monitoring functions that can trigger a stop in this range (e.g. monitoring spaces) must be deactivated beforehand. No safety functions may be executed that would trigger a stop (e.g. E-STOP, opening the safety gate, change of operating mode, etc.). If a brake has been identified as being defective, the parking position must be approached no faster than at 10% of maximum velocity.

WARNING

The parking position must be selected in a position where no persons are endangered if the robot sags because of the defective brake. The transport position, for example, can be selected as the parking position. Further information about the transport position is contained in the robot operating or assembly instructions.

WARNING

Make sure that the robot cannot collide and that no persons are in the motion range of the robot.

Precondition

- User rights of the following function groups:
  - **Jogging with the jog keys** or alternatively **Jogging using the 6D mouse**
  - **Teach local points**

  But at least the user group “**Expert**”
- All output signals are assigned to outputs.
- T1 mode

Procedure

Start position:

1. Open the program BrakeTestStart.src in the directory R1\TP\BrakeTest.

2. Teach the motions to the start position of the brake test.
   - The motions must be taught in such a way that the robot cannot cause a collision on the way to the start position.
   - In the start position, each robot axis to be tested must have a sufficient motion range.

     During the brake test, rotational axes move a maximum of 5° in the direction in which the software limit switch is situated further away; linear axes a maximum of 10 cm.
3. Save and close the program.

End position:

1. Open the program BrakeTestBack.src in the directory R1\TP\BrakeTest.
2. Teach the motions from the start position to the end position of the brake test.

   The start and end position may be identical.
3. Save and close the program.

Parking position:

1. Open the program BrakeTestPark.src in the directory R1\TP\BrakeTest.
2. Teach the motions from the end position to the parking position of the robot.
3. Save and close the program.

Alternatively, the parking position can also be taught later when testing the sequence.
(>>> *6.22.4.6 "Testing the sequence in the case of defective brakes" Page 282*)

6.22.4.6 Testing the sequence in the case of defective brakes

WARNING

If a brake is identified as being defective and the drives are deactivated, the robot may sag. For this reason, no stop may be triggered during the motion to the parking position. The monitoring functions that can trigger a stop in this range (e.g. monitoring spaces) must be deactivated beforehand. No safety functions may be executed that would trigger a stop (e.g. E-STOP, opening the safety gate, change of operating mode, etc.). If a brake has been identified as being defective, the parking position must be approached no faster than at 10% of maximum velocity.

WARNING

The parking position must be selected in a position where no persons are endangered if the robot sags because of the defective brake. The transport position, for example, can be selected as the parking position. Further information about the transport position is contained in the robot operating or assembly instructions.

WARNING

Make sure that the robot cannot collide and that no persons are in the motion range of the robot.

Description

The robot controller simulates a defective brake.

Precondition

- User rights: Function group **Program selection and deselection**
  But at least the user group “**Expert**”
- In the start position, an adequate motion range is available for each axis to be tested. (Or, if no start position has been taught, in the actual position.)
  During the brake test, rotational axes move a maximum of 5° in the direction in which the software limit switch is situated further away; linear axes a maximum of 10 cm.

Procedure

1. Select the program **BrakeTestReq.src** in the directory R1\TP\BrakeTest.
   The following message is displayed: *Start key required*
2. Press the Start key.
   The following message is displayed: **Should the brake test be carried out manually or should the movement to the parking position be checked?**
3. Select the answer **Park pos.**.
4. The BCO run is performed. The message *Programmed path reached (BCO)* is displayed.
5. Press the Start key.
   The following message is displayed: *CAUTION! Braking effect of the holding brakes no longer sufficient to hold the robot safely. Triggering safety functions or switching off the drives can cause the robot to sag. Following confirmation, the parking position/wait position is addressed.*.
   Confirm the message with **Park pos.**.
   - The robot moves to the parking position if the parking position has already been taught.
   - If the parking position has not yet been taught, the following message is displayed:
     Parking position is invalid. Move the robot to the correct position and press the “Touch Up” softkey.
     In this case, move the robot manually to the desired parking position and teach the position.

6.22.5 Performing a brake test

6.22.5.1 Performing a brake test for requested axes (cyclically via program)

Description

With BrakeTestReq.src, the axes for which there is a brake test request can be tested. The axes can either be tested in a single cycle or the test can be divided into several shorter cycles.

This allows, for example, small breaks in the application to be utilized for testing individual axes.

Precondition

To integrate the program:

- User rights: Function group **Critical KRL program changes**

Procedure

Test in a single cycle:

Integrate BrakeTestReq.src into a suitable program (e.g. into CELL.src or into the application program) in such a way that it is called cyclically as a subprogram. If a brake test is requested, BrakeTestReq.src detects the request and starts the brake test.

- Call BrakeTestReq.src without parameters:

```
BrakeTestReq()
```

Further information:

The axes are tested in a single cycle, from the lowest axis number to the highest.

Alternative procedure

Test divided into several shorter cycles:

Integrate BrakeTestReq.src into a suitable program in such a way that it is called cyclically as a subprogram. If a brake test is requested, BrakeTestReq.src detects the request and starts the brake test.

- When calling BrakeTestReq.src, transfer the axes to be tested as parameters.

Further information:

- One or more of the requested axes can be tested per cycle. The individual cycles do not have to follow one another directly.
- The order of the cycles is irrelevant. A cycle to test A6, for example, can be called first. If multiple axes are tested in one cycle, however, the lowest axis will always be tested first.
- In total, all requested axes must be tested. This must occur within the monitoring time.

Parameter

The axes to be tested can be transferred as an integer or as a bit mask.

Bit 0 corresponds to A1; bit 1 to A2; …; bit 6 to A7/E1; …; bit 11 to A12/E6.

Example for A1 and A3, as an integer:

```
BrakeTestReq(5)
```

Example for A1 and A3, as a bit mask:

```
BrakeTestReq('b101')
```

No parameter or parameter “-1” means: all active axes.

Example

In this example, the lowest numbered axis from amongst the axes with a currently requested brake test is tested. Such an example can be integrated into the application cycle at a suitable point in order to test one axis per cycle.

```
   int axes_bit_mask, test_bit_mask, counter
   ...
1 axes_bit_mask = get_axesmask(#braketest_required)
2 test_bit_mask = 1
3 if axes_bit_mask > 0 then
```

```
 4    for counter=1 to 12
 5      if (axes_bit_mask B_AND test_bit_mask) > 0 THEN
 6          counter = 13
 7      else
 8          test_bit_mask = test_bit_mask*2
 9      endif
10    endfor
11    braketestreq(test_bit_mask)
12 endif
   ...
```

Line	Description
1	Query for which axes the brake test is currently requested.
2	By way of preparation, set the lowest possible axis (in this case, A1) as the axis to be tested.
3 to 12	If there is a request for at least one axis, the IF block is executed. It contains a counting loop.
4 to 10	The counting loop is designed in such a way that, upon exiting it, `test_bit_mask` always corresponds to the lowest axis of all those to be tested.
6	Exit the counting loop.
11	Call the brake test for the axis to be tested `test_bit_mask`.

6.22.5.2 Performing a brake test for active axes (manually)

WARNING

If a brake is identified as being defective and the drives are deactivated, the robot may sag. For this reason, no stop may be triggered during the motion to the parking position. The monitoring functions that can trigger a stop in this range (e.g. monitoring spaces) must be deactivated beforehand. No safety functions may be executed that would trigger a stop (e.g. E-STOP, opening the safety gate, change of operating mode, etc.). If a brake has been identified as being defective, the parking position must be approached no faster than at 10% of maximum velocity.

WARNING

Make sure that the robot cannot collide and that no persons are in the motion range of the robot.

Description

This procedure can be used to test the active axes. The axes are tested in a single cycle, from the lowest axis number to the highest.

This procedure can be used to process an existing brake test request.

Precondition

- User rights: Function group **Program selection and deselection**
  But at least the user group “**Expert**”

- In the start position, an adequate motion range is available for each axis to be tested. (Or, if no start position has been taught, in the actual position.)

  During the brake test, rotational axes move a maximum of 5° in the direction in which the software limit switch is situated further away; linear axes a maximum of 10 cm.

Procedure

1. Select the program **BrakeTestReq.src** in the directory R1\TP\BrakeTest.

   The following message is displayed: *Start key required*
2. Press the Start key.

   The following message is displayed: **Should the brake test be carried out manually or should the movement to the parking position be checked?**
3. Select the answer **BT man.**.
4. The BCO run is performed. The message *Programmed path reached (BCO)* is displayed.
5. Press the Start key.

   The following message is displayed: *Brake test for axes {Axis bit mask} required*. The message lists the active axes.

   The program now tests the brakes/axes successively, starting with the lowest axis number.
6. Possible results:
   - If a brake is OK, this is indicated by the following message: *Brake {Brake no.}{Axis no.} OK*.

     The message *Brake test for axes {Axis bit mask} required* then reappears. It now only lists the axes that have not yet been tested.

     The program automatically tests the next axis.

     If all brakes are OK, this is indicated after the brake test by the following message: *Brake test successful*. (It is possible that one or more brakes may have reached the wear limit. This is also indicated by a message.)

     Now deselect the program BrakeTestReq.src.
   - If a brake is defective, this is indicated by the following message: *Insufficient holding torque of brake {Brake no.}{Axis no.}*. The test continues until all brakes have been tested.

     Once all brakes have been tested, the following message is displayed: *CAUTION! Braking effect of the holding brakes no longer sufficient to hold the robot safely. Triggering safety functions or switching off the drives can cause the robot to sag. Following confirmation, the parking position/wait position is addressed.*.

     Now press **Park pos.** to move the robot to the parking position.

6.22.5.3 Performing a brake test for further axes (e.g. couplable axes)

Description

With BrakeTestAxes.src, axes for which there is no brake test request can be tested. In particular, it also enables the testing of axes which cannot be activated for the brake test and thus cannot be tested via BrakeTestReq(). Couplable axes fall into this category, for example.

Precondition

To integrate the program:

- User rights: Function group **Critical KRL program changes**

Procedure

Integrate BrakeTestAxes.src into a suitable program in such a way that it is called as a subprogram.

- When calling BrakeTestAxes.src, transfer the axes to be tested as parameters.

Parameters

The axes to be tested can be transferred as an integer or as a bit mask.

Bit 0 corresponds to A1; bit 1 to A2; …; bit 6 to A7/E1; …; bit 11 to A12/E6.

Example for A7/E1, as an integer:

```
BrakeTestReq(64)
```

Example for A7/E1, as a bit mask:

```
BrakeTestReq('b1000000')
```

No parameter or parameter “-1” means: all active axes.

Additional info

In principle, it is also possible to use BrakeTestAxes(*axes*) to test one or more axes for which there is a request. When calling the test, the axes must be transferred as parameters in this case too.

An existing brake test request can be processed using BrakeTestAxes.src.

> If BrakeTestAxes.src is called for an active axis, this triggers a request for all active axes! As usual, the request must be processed within the monitoring time. It is therefore not possible to test individual active axes using BrakeTestAxes.src.

6.22.6 Automatic brake check

Description

The default cycle time of the brake test is 46 hours. In order to detect defective brakes as early as possible, however, even before the end of the cycle time, the robot controller performs an additional, automatic brake check.

If the brake check indicates that a brake might be defective, a brake test must be performed for verification.

Decoupled axes and force-controlled axes are excluded from the brake check.

Sequence

1. If a brake has been applied, the robot controller automatically checks whether the axis is still moving. Motions within a narrow, internally defined tolerance range are allowed.

> The tolerance range corresponds to half of the standstill window. The standstill window is: $IN_POS_MA[axis] * $IN_STILL_MA. Further information about these system variables can be found in the documentation **Configuration of Kinematic Systems**.

2. If the tolerance range is exceeded, this indicates that the brake is defective. In this case, the robot controller switches the axis back to servo control to prevent it from sagging. The axis is now in the “BrakeDefect” state.

   Furthermore, the status message *Brake defective, {Axis} permanently under servo control* is displayed.
3. The further procedure depends on whether or not the brake test is active for the corresponding axis:
   - **Brake test is active:**

     The robot controller sets $BRAKETEST_REQ_INT to TRUE and generates the following message: *Brake test required*. A brake test must be performed within the next 2 hours, otherwise the robot will stop!
   - **Brake test is not active:**

     The robot stops and the acknowledgement message *Stop due to defective brake* is displayed.

     The message can be acknowledged and the robot movement can be resumed. The axis remains permanently under servo control, i.e. its brakes are no longer applied. (Except if the safety controller resets the “Motion enable” input.)

     In order to enable the brakes to close again, a brake test must be performed manually!
4. Once a brake test has been performed and has indicated that the axis is working correctly, the status message *Brake defective, {Axis} permanently under servo control* disappears again.

BrakeDefect

If an axis is in the “BrakeDefect” state, the robot controller ignores brake closing requests for this axis and for the axes on the same brake channel.

If an axis is in the “BrakeDefect” state, its brakes nevertheless close in the following case: the safety controller resets the “Motion enable” input, e.g. in the case of an EMERGENCY STOP or short-circuit braking.

6.22.7 System functions for the brake test

6.22.7.1 GET_AXESMASK()

Description

Various queries concerning the axes involved in the brake test can be carried out.

Syntax

result = `GET_AXESMASK(`*axes*`)`

Explanation of the syntax

Element	Description
result	Variable for the return value, type: INT Bit mask, i.e. specification of which axes are involved
axes	Type: ENUM AXESMASK_INFO • #BRAKETEST_CONFIGURED Axes configured for the brake test Corresponds to the axes in the **Active Configuration** column in the **Brake test configuration** window. • #BRAKETEST_ACTIVATED – If the brake test is active, the return value is as for #BRAKETEST_CONFIGURED. – If the brake test is not active, the return value is “0”. • #BRAKETEST_REQUIRED Axes for which the brake test is currently requested • #BRAKETEST_UNTESTED Axes for which the state of the brake is BT_UNTESTED. Axes not configured for the brake test are also included. • #BRAKETEST_BRAKES_OK Axes found to be OK in the most recent brake test

Example

Query for which axis the brake test is currently requested:

```
int axes_bit_mask
...
axes_bit_mask = get_axesmask(#braketest_required)
```

6.22.7.2 GET_BRAKETEST_TIME()

Description

Various time values related to the brake test can be polled.

Syntax

result = `GET_BRAKETEST_TIME(`*time_type*`)`

Explanation of the syntax

Element	Description
result	Variable for the return value, type: REAL Time (unit: h)
time_type	Times that can be polled Type: ENUM BRAKETEST_TIME_INFO • #BT_CONFIG_CYCLE_TIME Cycle time for the brake test Corresponds to the **Cycle time [h]** box in the **Brake test configuration** window. **Cycle time [h]** • #BT_REMAINING_CYCLE_TIME Remaining cycle time • #BT_REMAINING_MON_TIME Remaining monitoring time

7 Program and project management

7.1 Creating a new program

Precondition

- The Navigator is displayed.

Procedure

1. In the directory structure, select the folder in which the program is to be created, e.g. the **Program** folder. (Not all folders allow the creation of programs within them.)
2. Press **New**.
3. Only in the user group “Expert” or higher:

   The **Template selection** window opens. Select the desired template and confirm with **OK**.
4. Enter a name for the program and confirm it with **OK**.

7.2 Creating a new folder

Precondition

- The Navigator is displayed.

Procedure

1. In the directory structure, select the folder in which the new folder is to be created, e.g. the folder **R1**.

   Not all folders allow the creation of new folders within them. In the user groups “Operator” and “User”, new folders can only be created in the folder **R1**.
2. Press **New**.
3. Enter a name for the folder and confirm it with **OK**.

7.3 Renaming a file or folder

Precondition

- The Navigator is displayed.

Procedure

1. In the directory structure, select the folder in which the file or folder to be renamed is located.
2. Select the file or folder in the file list.
3. Select **Edit** > **Rename**.
4. Overwrite the name with the new name and confirm with **OK**.

7.4 Navigator file manager

Overview

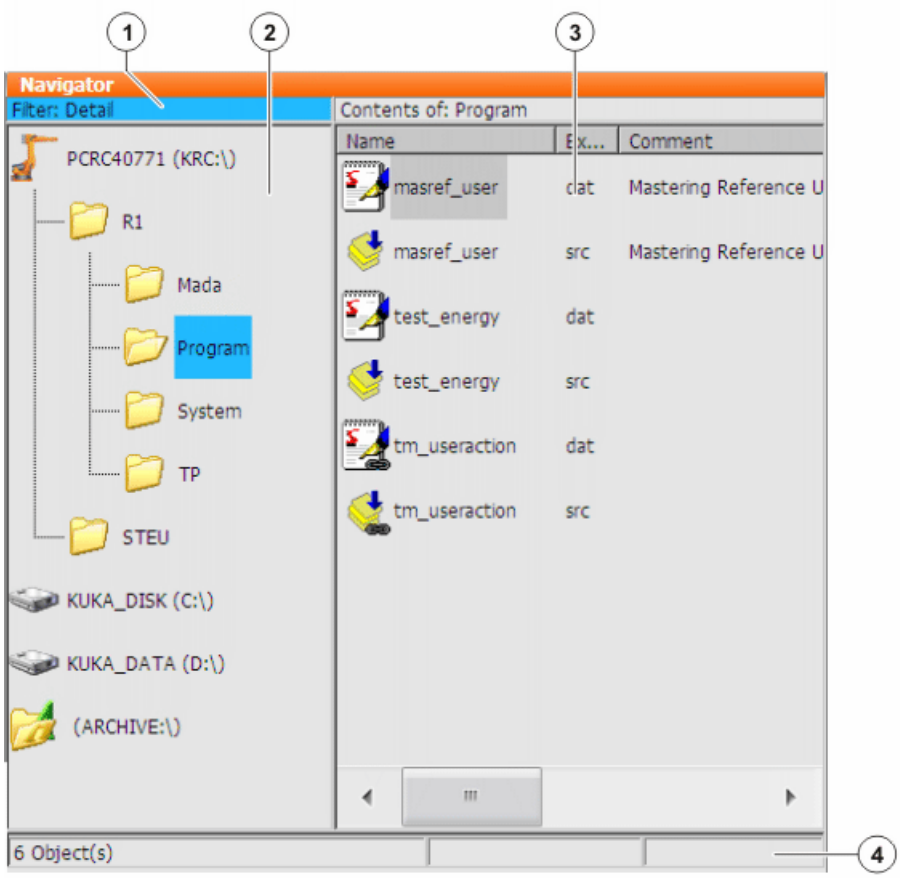


Fig. 7-1: Navigator

1	Header	3	File list
2	Directory structure	4	Status bar

Description

In the Navigator, the user manages programs and system-specific files.

Header

- Left-hand area: the selected filter is displayed.
  (>>> *7.4.1 "Selecting filters" Page 293*)
- Right-hand area: the directory or drive selected in the directory structure is displayed.

Directory structure

Overview of directories and drives. Exactly which directories and drives are displayed depends on the user group and configuration.

File list

The contents of the directory or drive selected in the directory structure are displayed. The manner in which programs are displayed depends on the selected filter.

The file list has the following columns:

Column	Description
Name	Directory or file name
Extension	File extension This column is not displayed in the user groups “Operator” and “User”.
Comment	Comment
Attribute	Attributes of the operating system and kernel system This column is not displayed in the user groups “Operator” and “User”.
Size	File size in kilobytes This column is not displayed in the user groups “Operator” and “User”.
#	Number of changes made to the file
Changed	Date and time of the last modification
Created	Date and time of file creation This column is not displayed in the user groups “Operator” and “User”.

Status bar

The status bar can display the following information:

- Selected objects
- Action in progress
- User dialogs
- User entry prompts
- Requests for confirmation

7.4.1 Selecting filters

Description

The filter defines how programs are displayed in the file list for the user group “Expert” or higher. The following filters are available:

- **Detail**: Programs are displayed as SRC and DAT files. (Default setting)
- **Module**: Programs are displayed as modules.

For the user groups “Operator” and “User”, programs are always displayed as modules.

Procedure

1. Select the menu sequence **Edit** > **Filter**.
2. Select the desired filter in the left-hand section of the Navigator.
3. Confirm with **OK**.

7.4.2 Displaying or modifying properties of files and folders

Precondition

- To change properties:
  User rights: Function group **Critical KRL program changes**

Procedure

1. Select the object in the directory structure or in the file list.
2. Select the menu sequence **Edit** > **Properties**.
   A window opens. Depending on the specific object selected, the number of tabs in the window may vary.
3. If required: Change the properties and save the changes with **OK**.

General

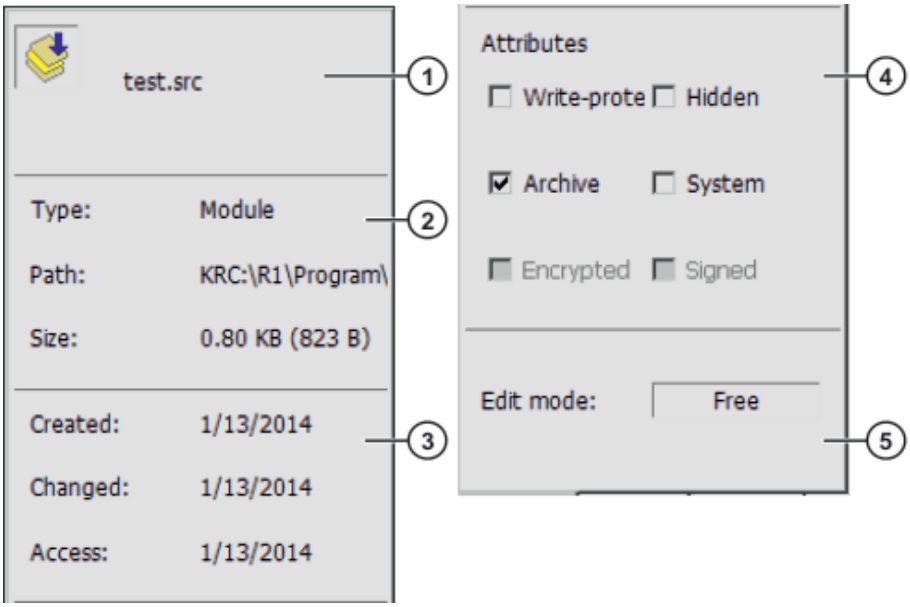


Fig. 7-2: “General” tab

Item	Description
1	Name of the selected object
2	Object type, path and size. Object types: • **Module**: Module • **Dir**: Folder • **Archive**: Archive file • **Bin**: Binary file • **Text**: Text file • **VirtualDir**: Virtual folder • **Unknown**: All other file types
3	Windows object properties
4	Windows object properties
5	**Free**: The file is not selected on the smart.HMI and is not open. **Full**: The file is open on the smart.HMI. **ProKor**: The file is selected on the smart.HMI.

Module info

The **Module info** tab is only displayed if the selected object is a file.

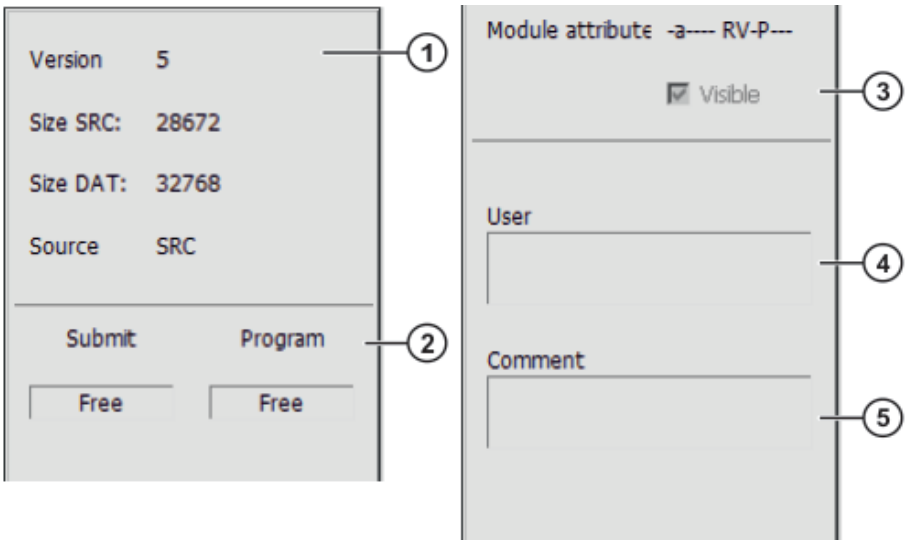


Fig. 7-3: “Module info” tab

Item	Description
1	**Version**: internal version number of the file. After creation, the file does not yet have a number. After the first change, the file receives the number 1. The number is incremented after every change. **Size SRC:**: size of the SRC file **Size DAT:**: size of the DAT file **Source type:**: file type • **SRC**: SRC file • **SubmitSub**: SUB file • **None**: all other file types, e.g. DAT file
2	Status of the module in the submit interpreter and in the robot interpreter **Free**: program is not selected. **selected**: program is selected. **Active**: only relevant for the **Submit** box. This program is currently being used by the submit interpreter.
3	Check box active: if this program is called as a subprogram, it is displayed in the Editor. Check box not active: if this program is called as a subprogram, it is not displayed in the Editor. This program cannot be selected manually.
4	The user can enter his or her name here.
5	The user can enter a comment for the module here. The comment is displayed in the **Comment** column in the Navigator.

Parameters

The **Parameters** tab is only displayed if the selected object is a file.

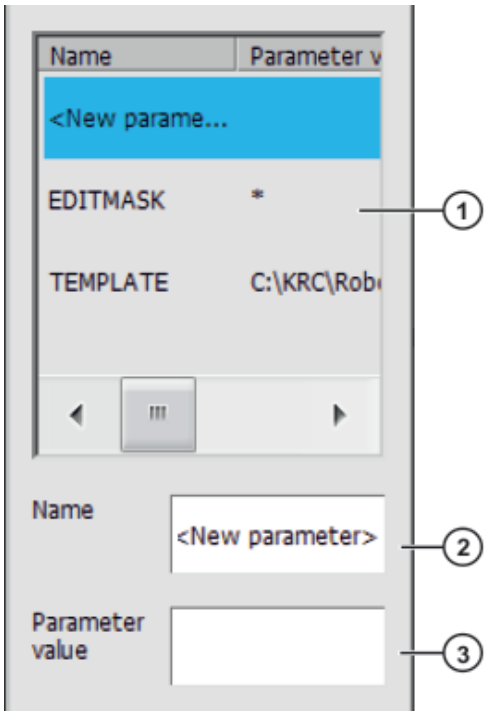


Fig. 7-4: “Parameters” tab

Any desired information can be stored in KRL modules.

Item	Description
1	The existing information is shown here. A piece of information can be deleted by selecting the line and deleting the contents in the box **Parameter value**. Then confirm with **OK**.
2	The user can enter a name here for a new piece of information.
3	The user can enter information here.

Program in the editor

If an SRC or DAT file is opened in an editor (e.g. WordPad) in Windows, a number of the file properties are displayed above the DEF line.

```
1 &ACCESS RV
2 &REL 2
3 &COMMENT test comment
4 &USER kuka
5 &PARAM test name = test param
6 &PARAM TEMPLATE = C:\KRC\Roboter\Template\vorgabe
7 &PARAM EDITMASK = *
8 DEF test( )
...
```

Line	Description
1	Tab **Module info**, check box **Visible** • `&ACCESS RV` = check box active • `&ACCESS R` = check box inactive
2	Tab **Module info**, box **Version**
3	Tab **Module info**, box **Comment**

Line	Description
4	Tab **Module info**, box **User**
5	Tab **Parameters**, boxes **Name** and **Parameter value**

7.5 Selecting or opening a program

Overview

A program can be selected or opened. Instead of the Navigator, an editor is then displayed with the program.

It is possible to toggle backwards and forwards between the program display and the Navigator.

Differences

Program is selected:

- The block pointer is displayed.
- The program can be started.
- The program can be edited to a certain extent.

  Example: KRL instructions covering several lines (e.g. LOOP … ENDLOOP) are not permissible.
- When the program is deselected, modifications are accepted without a request for confirmation. If impermissible modifications are programmed, an error message is displayed.

Program is open:

- The program cannot be started.
- The program can be edited.
- A request for confirmation is generated when the program is closed. Modifications can be accepted or rejected.

7.5.1 Selecting and deselecting a program

Precondition

- T1, T2 or AUT mode

Procedure

1. Select the program in the Navigator and press **Select**.

   The program is displayed in the editor. It is irrelevant whether a module, an SRC file or a DAT file is selected. It is always the SRC file that is displayed in the editor.
2. Start or edit the program.
3. Deselect the program again:

   Select **Edit** > **Cancel program**.

   or: In the status bar, touch the **Robot interpreter** status indicator. A window opens. Select **Cancel program**.

> When the program is deselected, modifications are accepted without a request for confirmation!

If the program is running, it must be stopped before it can be deselected.

Description

If a program is selected, this is indicated by the **Robot interpreter** status indicator.

(>>> *8.6 "Robot interpreter status indicator" Page 332*)

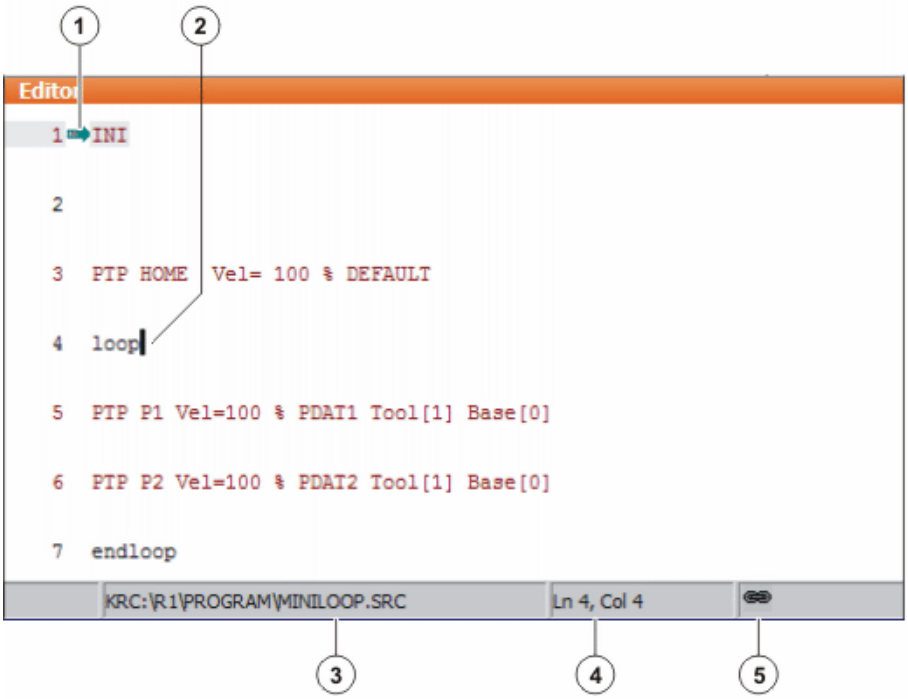


Fig. 7-5: Program is selected

1 Block pointer
2 Cursor
3 Program path and file name
4 Position of the cursor in the program
5 The icon indicates that the program is selected.

7.5.2 Opening a program

Precondition

- T1, T2 or AUT mode

A program can be opened in AUT EXT mode, but not edited.

Procedure

1. Select the program in the Navigator and press **Open**. The program is displayed in the editor.

   If a module has been selected, the SRC file is displayed in the editor. If an SRC file or DAT file has been selected, the corresponding file is displayed in the editor.
2. Edit the program.
3. Close the program.
4. To accept the changes, answer the request for confirmation with **Yes**.

Description

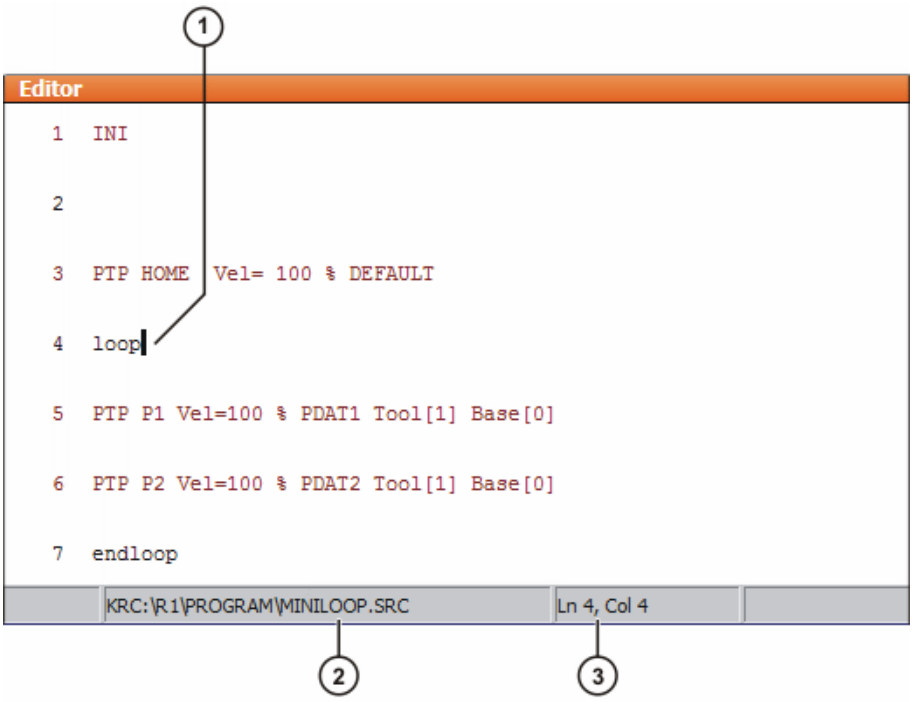


Fig. 7-6: Program is open

1 Cursor
2 Program path and file name
3 Position of the cursor in the program

7.5.3 Toggling between the Navigator and the program

Description

If a program is selected or open, it is possible to display the Navigator again without having to deselect or close the program. The user can then return to the program.

Procedure

Program is selected:

- Toggling from the program to the Navigator: select the menu sequence **Edit** > **Navigator**.
- Toggling from the Navigator to the program: press **PROGRAM**.

Program is open:

- Toggling from the program to the Navigator: select the menu sequence **Edit** > **Navigator**.
- Toggling from the Navigator to the program: press **EDITOR**.

Programs that are running or have been interrupted must first be stopped before the menu sequences and buttons referred to above are available.

7.6 Structure of a KRL program

```
1   DEF my_program( )
2   INI
3
4   PTP HOME Vel= 100 % DEFAULT
    ...
8   LIN point_5 CONT Vel= 2 m/s CPDAT1 Tool[3] Base[4]
    ...
14  PTP point_1 CONT Vel= 100 % PDAT1 Tool[3] Base[4]
    ...
20  PTP HOME Vel= 100 % DEFAULT
21
22  END
```

Line	Description
1	The DEF line indicates the name of the program. If the program is a function, the DEF line begins with “DEFFCT” and contains additional information. The DEF line can be displayed or hidden. (>>> *7.7.1 "Displaying/hiding the DEF line" Page 301*)
2	The INI line contains initializations for internal variables and parameters.
4	HOME position (>>> *7.6.1 "HOME position" Page 300*)
8	LIN motion
14	PTP motion
20	HOME position
22	The END line is the last line in any program. If the program is a function, the wording of the END line is “ENDFCT”. The END line must not be deleted!

The first motion instruction in a KRL program must define an unambiguous starting position. The HOME position, which is stored by default in the robot controller, ensures that this is the case.

If the first motion instruction is not the default HOME position, or if this position has been changed, one of the following statements must be used:

- Complete PTP instruction of type POS or E6POS
- Complete PTP instruction of type AXIS or E6AXIS

“Complete” means that all components of the end point must be specified.

WARNING

If the HOME position is modified, this affects all programs in which it is used. Injuries or damage to property may result.

In programs that are used exclusively as subprograms, different statements can be used as the first motion instruction.

7.6.1 HOME position

The HOME position is not program-specific. It is generally used as the first and last position in the program as it is uniquely defined and uncritical.

The HOME position is stored by default with the following values in the robot controller:

Axis	A1	A2	A3	A4	A5	A6
Pos.	0°	- 90°	+ 90°	0°	0°	0°

Additional HOME positions can be taught. A HOME position must meet the following conditions:

- Good starting position for program execution
- Good standstill position. For example, the stationary robot must not be an obstacle.

WARNING

If the HOME position is modified, this affects all programs in which it is used. Injuries or damage to property may result.

7.7 Displaying/hiding program sections

7.7.1 Displaying/hiding the DEF line

Description

By default, the DEF line is hidden. Declarations can only be made in a program if the DEF line is visible.

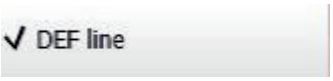
The DEF line is displayed and hidden separately for opened and selected programs. If detail view (ASCII mode) is activated, the DEF line is visible and does not need to be activated separately.

Precondition

- User rights: Function group **Critical KRL program changes**
- Program is selected or open.

Procedure

1. Select the menu sequence **Edit** > **View**. The subitem **DEF line** displays the current status:
   - **Check box not active**: The DEF line is hidden.
   - **Check box active**: The DEF line is displayed.

2. To change the status, touch the menu item **DEF line**.

   The menu then closes automatically.

7.7.2 Activating detail view

Description

Detail view (ASCII mode) is deactivated by default to keep the program transparent. If detail view is activated, hidden program lines, such as the FOLD and ENDFOLD lines and the DEF line, are displayed.

Detail view is activated and deactivated separately for opened and selected programs.

Precondition

- User rights: Function group **Critical KRL program changes**

Procedure

1. Select the menu sequence **Edit** > **View**. The subitem **Detail view (ASCII)** displays the current status:
   - **Check box not active**: Detail view is deactivated.
   - **Check box active**: Detail view is activated.

2. To change the status, touch the menu item **Detail view (ASCII)**.
   The menu then closes automatically.

7.7.3 Activating/deactivating the line break function

Description

If a line is wider than the program window, the line is broken by default. The part of the line after the break has no line number and is marked with a black, L-shaped arrow.

```
8  EXT  IBGN (IBGN_COMMAND  :IN,BOOL  :IN,REAL  :IN,REAL
  ↳ :IN,BOOL  :IN,E6POS  :OUT )
```

Fig. 7-7: Line break

The line break function can be deactivated. If a line is wider than the program window, the line is no longer visible in its entirety. A scroll bar is displayed underneath the program window.

The line break function is activated and deactivated separately for opened and selected programs.

Precondition

- Program is selected or open.

Procedure

1. Select the menu sequence **Edit** > **View**. The subitem **Line break** displays the current status:
   - **Check box not active**: Line break function is deactivated.
   - **Check box active**: Line break function is activated.

2. To change the status, touch the menu item **Line break**.
   The menu then closes automatically.

7.7.4 Displaying folds

Description

Folds are used to hide sections of the program. In this way, folds make programs more transparent. The hidden program sections are processed

during program execution in exactly the same way as normal program sections.

If a program is deselected, all folds are automatically closed.

```
2

3  PTP HOME  Vel= 100 % DEFAULT

4
```

Fig. 7-8: Example of a closed Fold

```
2

3  PTP HOME  Vel= 100 % DEFAULT

4  $BWDSTART = FALSE

5  PDAT_ACT=PDEFAULT

6  FDAT_ACT=FHOME

7  BAS (#PTP_PARAMS,100 )

8  $H_POS=XHOME

9  PTP  XHOME

10
```

Fig. 7-9: Example of an open Fold

Color coding of Folds:

Color	Description
Dark red	Closed fold
Light red	Opened fold
Dark blue	Closed sub-Fold
Light blue	Opened sub-Fold
Green	Contents of the Fold

Precondition

- Program is selected or open.

Procedure

1. Select the line containing the Fold.
2. Press **Open/close fold**. The Fold then opens.
3. To close the fold, press **Open/close fold** again.

Alternatively, use the menu sequence **Edit** > **FOLD** > **Open all FOLDs** or **Close all FOLDs** to open or close all the Folds in a program at once.

7.8 Editing programs

7.8.1 Inserting a comment or stamp

Precondition

- Program is selected or open.
- Operating mode T1

Procedure

1. Select the line after which the comment or stamp is to be inserted.
2. Select the menu sequence **Commands** > **Comment** > **Normal** or **Stamp**.
3. Enter the desired data. If a comment or stamp has already been entered previously, the inline form still contains the same entries.
   - In the case of a comment, the box can be cleared using **New text** ready for entry of a new text.
   - In the case of a stamp, the system time can also be updated using **New time** and the **NAME** box can be cleared using **New name**.
4. Save with **Cmd Ok**.

Description Comment

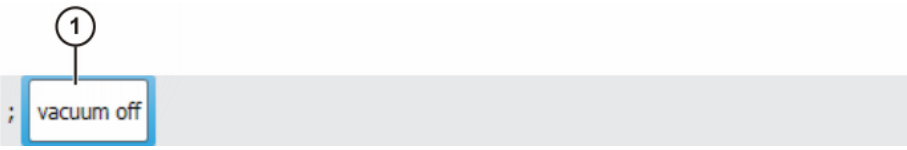


Fig. 7-10: Inline form “Comment”

Item	Description
1	Any text

Description Stamp

A stamp is a comment that is extended to include the system date and time and the user ID.

Fig. 7-11: Inline form “Stamp”

Item	Description
1	System date (cannot be edited)
2	System time
3	Name or ID of the user
4	Any text

7.8.2 Selecting a range

Precondition

- User rights: Function group **General KRL program changes**
- Program is selected or open.

Procedure

1. Touch the line in which the selection is to start.
2. Select the menu sequence **Edit** > **Mark region**.
   The line in which the cursor is positioned is now selected.
3. Touch the line in which the selection is to end.
   The range between the lines, including the start and end lines, is now selected.

7.8.3 Deleting program lines

Lines cannot be restored once they have been deleted!

Description

If a program line containing a motion instruction is deleted, the point name and coordinates remain saved in the DAT file. The point can be used in other motion instructions and does not need to be taught again.

Precondition

- Program is selected or open.
- Operating mode T1

Procedure

1. Select the line to be deleted. (The line need not have a colored background. It is sufficient for the cursor to be in the line.)
   If several consecutive lines are to be deleted: drag a finger or stylus across the desired area. (The area must now have a colored background.)
2. Select the menu sequence **Edit** > **Delete**.
3. Confirm the request for confirmation with **Yes**.

7.8.4 Creating folds

Syntax

`;FOLD` *Name*

Statements

`;ENDFOLD` *<Name>*

The ENDFOLD lines can be assigned more easily if the name of the Fold is entered here as well. Folds can be nested.

Precondition

- User group “Expert” or higher
- Program is selected or open.
- T1 mode

Procedure

1. Enter Fold in program. A double semicolon prevents the Fold from closing when edited.

```
4
5  ;;FOLD outputs
6  $OUT[1] = TRUE
7  $OUT[2] = TRUE
8  ;;ENDFOLD outputs
9
```

Fig. 7-12: Creating a sample Fold, step 1

2. Delete the second semicolon.

```
4
5  ;FOLD outputs
6  $OUT[1] = TRUE
7  $OUT[2] = TRUE
8  ;ENDFOLD outputs
9
```

Fig. 7-13: Creating a sample Fold, step 2

3. Position the cursor in a line outside the Fold. The Fold closes.

```
4
5  outputs
6
```

Fig. 7-14: Creating a sample Fold, step 3

7.8.5 Additional editing functions

The following additional program editing functions can be called using **Edit**:

Cut, **Copy**, **Paste**

Precondition:

- User rights: Function group **General KRL program changes**
- Program is selected or open.
- T1 mode

Find

Precondition:

- Program is selected or open.

Replace

Precondition:

- User rights: Function group **Critical KRL program changes**
- Program has been opened.
- T1 mode

Marked region

(>>> *10.6.4 "Transforming blocks of coordinates" Page 426*)

7.9 Printing a program

Precondition

- User rights: Function group **Print functions**

Procedure

1. Select the program in the Navigator. Multiple program selection is also possible.
2. Select the menu sequence **Edit** > **Print**.

7.10 Archiving and restoring data

7.10.1 Archiving overview

Target locations

Archiving can be performed to the following target destinations:

- USB stick in smartPAD or robot controller
- Network

Menu items

The following menu items are available:
("*.*" means all files and subdirectories.)

Menu item	Archives the directories/files
All	• KRC:*.* • C:\KRC\Roboter\Config\User*.* • C:\KRC\Roboter\Config\System\Common\Mada*.* • C:\KRC\Roboter\Template*.* • C:\KRC\Roboter\Rdc*.* • C:\KRC\User*.* • Some additional log data Registry entries are also archived.
Applications	• KRC:\R1\Program*.* • KRC:\R1\System*.* • KRC:\R1\cell*.* • KRC:\Steu\$config*.*

Menu item	Archives the directories/files
System data	• KRC:\R1\Mada*.* • KRC:\R1\System*.* • KRC:\R1\TP*.* • KRC:\Steu\Mada*.* • C:\KRC\Roboter\Config\User*.* • C:\KRC\Roboter\Config\System\Common\Mada*.* • C:\KRC\Roboter\Template*.* • C:\KRC\Roboter\Rdc*.* • C:\KRC\User*.* Registry entries are also archived.
Log data	• C:\KRC\Roboter\log*.* Except: Poslog.xsl and files with the extension DMP • Some additional log data
KrcDiag	If it is necessary for an error to be analyzed by KUKA Deutschland GmbH, this menu item can be used to compress the data for sending to KUKA. A screenshot of the current view of the smartHMI is automatically generated for the data packet. For this reason, display error-relevant information on the smartHMI before starting the procedure: For example, expand the message window or display the logbook. What information is useful here depends on the specific circumstances. In addition to the menu sequence **File** > **Archive**, there are other methods available for compressing these data. (>>> *13.5 "Automatically compressing data for error analysis (KRCDiag)" Page 599*)

If archiving is carried out using the menu item **All** and there is an existing archive present, this will be overwritten.

If archiving is carried out using a menu item other than **All** or **KrcDiag** and an archive is already available, the robot controller compares its name with that in the archive. If the names are different, a request for confirmation is generated.

If archiving is carried out repeatedly via **KrcDiag**, a maximum of 10 archives can be created. Further archives will overwrite the oldest existing archive.

The logbook can also be activated.

7.10.2 Archiving to a USB stick

Description

This procedure generates a ZIP file on the stick. By default, this file has the same name as the robot. A different name can be defined for the file, however, under **Start-up** > **Robot data**.

The archive is displayed in the ARCHIVE:\ directory in the Navigator. Archiving is also carried out automatically to D:\ as well as to the stick. The file INTERN.ZIP is generated here.

Special case **KrcDiag**:

This menu item generates the folder **KRCDiag** on the stick. This contains a ZIP file. The ZIP file is also automatically archived in C:\KUKA \KRCDiag.

NOTICE
A non-bootable USB stick must be used. We recommend using a non-bootable KUKA stick. Data may be lost if a stick from a different manufacturer is used.

Precondition

- For the subitem **All**:

  User rights of the function group **Archive to USB drives**
- For the subitems **Applications**, **System data**, **Log data**:

  User rights of the following function groups:
  - **Archive to USB drives**
  - **Partial archiving**

Procedure

1. Connect the USB stick (to smartPAD or cabinet).
2. In the main menu, select **File** > **Archive** > **USB (KCP)** or **USB (cabinet)** and then the desired menu item.
3. Confirm the request for confirmation with **Yes**. The archive is created.

   Once the archiving is completed, this is indicated in the message window.

   Special case **KrcDiag**: If archiving is carried out using this menu item, a separate window indicates when archiving has been completed. The window is then automatically hidden again.
4. The stick can now be removed.

7.10.3 Archiving on the network

Description

This procedure generates a ZIP file on the network path. By default, this file has the same name as the robot. A different name can be defined for the file, however, under **Start-up** > **Robot data**.

The network path to which the data are to be archived must be configured under **Start-up** > **Robot data**. If a user name and password are required for archiving to this path, these can also be entered here.

The archive is displayed in the ARCHIVE:\ directory in the Navigator. Archiving is also carried out automatically to D:\ as well as to the network path. The file INTERN.ZIP is generated here.

Special case **KrcDiag**:

This menu item generates the folder **KRCDiag** on the network path. This contains a ZIP file. The ZIP file is also automatically archived in C:\KUKA \KRCDiag.

Precondition

- The network path to which the data are to be archived is configured.
- For the subitem **All**:

  User rights of the function group **Archive to network**

- For the subitems **Applications**, **System data**, **Log data**:
  User rights of the following function groups:
  - **Archive to network**
  - **Partial archiving**

Procedure

1. In the main menu, select **File** > **Archive** > **Network** and then the desired menu item.
2. Confirm the request for confirmation with **Yes**. The archive is created.
   Once the archiving is completed, this is indicated in the message window. Special case **KrcDiag**: If archiving is carried out using this menu item, a separate window indicates when archiving has been completed. The window is then automatically hidden again.

7.10.4 Archiving the logbook

Description

The file Logbuch.txt is generated as an archive in the directory C:\KRC \ROBOTER\LOG.

Precondition

- User rights: Function group **Archive to local HDD/SSD**

Procedure

- In the main menu, select **File** > **Archive** > **Logbook**.
  The archive is created. Once the archiving is completed, this is indicated in the message window.

7.10.5 Restoring data

Description

WARNING

Only KSS 8.5 archives may be loaded into KSS 8.5. If other archives are loaded, the following may occur:
- Error messages
- Robot controller is not operable.
- Personal injury and damage to property.

If the archived files are not the same version as the files present in the system, an error message is generated during restoration.

Similarly, if the version of the archived technology packages does not match the installed version, an error message is generated.

Precondition

- For the subitem **All**:
  User rights of the function group **Restore**
- For the subitems **Applications** and **System data**:
  User rights of the function group **Partial restoration**

- If data are to be restored from the USB stick: A USB stick with the archive is connected.

  The stick can be connected to the smartPAD or robot controller.

NOTICE
A non-bootable USB stick must be used. We recommend using a non-bootable KUKA stick. Data may be lost if a stick from a different manufacturer is used.

Procedure

1. In the main menu, select **File** > **Restore** and then the desired subitems.
2. Confirm the request for confirmation with **Yes**. Archived files are restored to the robot controller. A message indicates completion of the restoration process.
3. If data have been restored from a USB stick: the stick can now be removed.
4. Reboot the robot controller.

7.11 Project management

7.11.1 Pinning a project on the robot controller

Description

Projects that are present on the robot controller can be pinned. A project can be pinned directly on the robot controller or in WorkVisual.

Pinned projects cannot be changed, activated or deleted. They can be copied or unpinned, however. A project can thus be pinned, e.g. to prevent it from being accidentally deleted.

Information about how projects can be pinned via WorkVisual can be found in the **WorkVisual** documentation.

Precondition

- User rights: Function group **General configuration**

Procedure

1. Touch the WorkVisual icon on the smartHMI, then go to **Open**. The **Project management** window opens.
2. Select the **Projects** tab.
3. Select the desired project and press the **Pin** button. The project is pinned and labeled with a pin symbol in the project list.
4. The project can be unpinned again by pressing the **Unpin** button.

7.11.2 Activating an existing project

Description

- A project that already exists on the robot controller can be activated there.
- A project can be transferred to the robot controller from WorkVisual and then activated.

Information about transferring and activating WorkVisual projects can be found in the **WorkVisual** documentation.

Precondition

- User group Expert

If the project to be activated contains changes in the safety settings, the precondition for activation is the user group “Safety maintenance”.
If a safety option is installed on the robot controller, e.g. KUKA.SafeOperation, different preconditions may apply. Information can be found in the documentation of the safety options.

Procedure

1. Touch the WorkVisual icon on the smartHMI, then go to **Open**. The **Project management** window opens.
2. Select the required tab:
   - **Projects**: All available projects are displayed here, including projects that have never yet been active. Different versions of the same project are not displayed.
     (>>> *7.11.3.1 "Projects tab" Page 313*)
   - **Restoration points**: The most recently active projects are displayed here. If a project has already been active more than once, the individual versions of the project are displayed.
     (>>> *7.11.3.2 "Restoration points tab" Page 315*)
3. Select and activate the desired project:
   - On the **Projects** tab, using the **Activate** button
   - On the **Restoration points** tab, using the **Restore** button
4. The KUKA smartHMI displays the request for confirmation *Do you want to activate the project […]?*. In addition, a message is displayed as to whether the activation would overwrite a project and, if so, which one.

   If no relevant project will be overwritten: Confirm with **Yes** within 30 minutes.
5. An overview is displayed of the changes which will be made in comparison to the project that is still active on the robot controller. The check box **Details** can be used to display details about the changes.

   WARNING

   If changes are listed in the overview under the heading **Safety-relevant communication parameters**, this means that the behavior of the Emergency Stop and “Operator safety” signal may have changed compared with the previous project.
   After activation of the project, the Emergency Stop and the “Operator safety” signal must be checked for safe functioning. If the project is activated on several robot controllers, this check must be carried out for every robot controller. Failure to carry out this check may result in death, injuries or damage to property.
6. The overview displays the request for confirmation *Do you want to continue?*. Confirm with **Yes**. The project is activated on the robot controller.

WARNING

After activation of a project on the robot controller, the safety configuration must be checked there! If this is not done, the robot will possibly be operated with incorrect data. Death, injuries or damage to property may result.
(>>> *6.6 "Checking the safety configuration of the robot controller" Page 215*)

WARNING

If the activation of a project fails, an error message is displayed. In this case, one of the following measures must be carried out:

- Either: Activate a project again (the same one or a different one).
- Or: Reboot the robot controller with a cold restart.

In the case of a KSS/VSS update, the initial project and base project are overwritten by copies of the active project.

7.11.3 Project management window

The **Project management** window is opened using the WorkVisual icon on the smartHMI.

7.11.3.1 Projects tab

Description

In addition to the regular projects, the **Project management** window contains the following special projects:

Project	Description
Initial project	The initial project is always present. It cannot be changed by the user. It contains the initial state of the robot controller as shipped.
Base project	The user can save the active project as the base project. This functionality is generally used to save a functional, tried-and-tested project state. The base project cannot be activated, but copied. The base project can no longer be changed by the user. It can, however, be overwritten by saving a new base project (after a request for confirmation). If a project is activated which does not contain all the configuration files, the missing information is inserted from the base project. This is the case, for example, if a project is activated from an earlier version of WorkVisual. (The configuration files include machine data files, safety configuration files and many others.)

In the case of a KSS/VSS update, the initial project and base project are overwritten by copies of the active project.

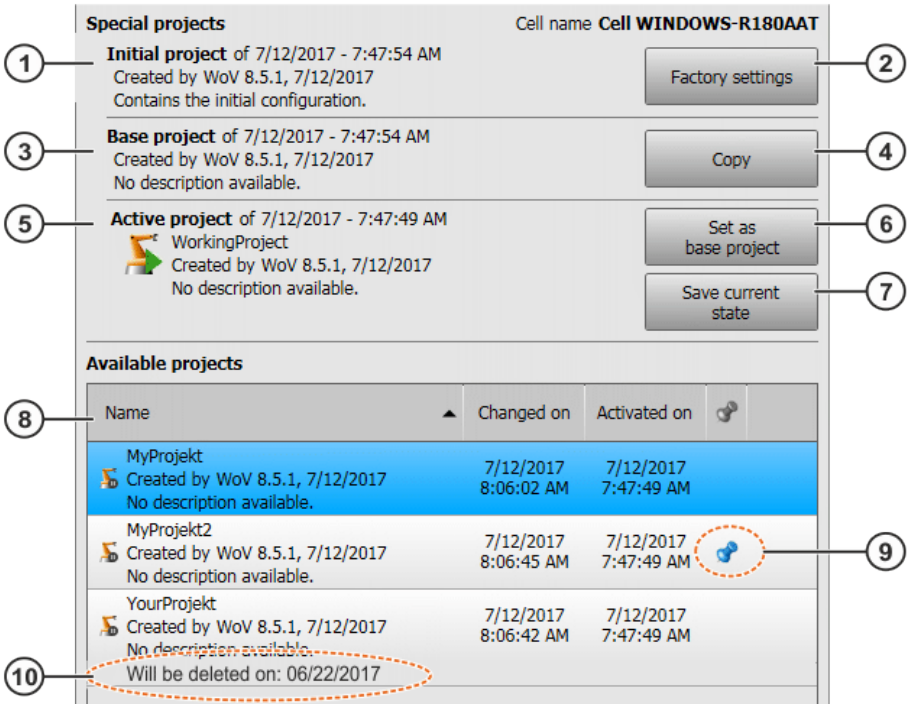


Fig. 7-15: Projects tab

Item	Description
Special projects	
1	The initial project is displayed.
2	Restores the factory settings of the robot controller. Required user rights: function group **Critical KRL program changes**
3	The base project is displayed.
4	Creates a copy of the base project.
5	The active project is displayed.
6	Saves the active project as the base project. The active project remains active and the previous base project is deleted. Required user rights: function group **General configuration**
7	Creates a pinned copy of the active project.
Available projects	
8	List of inactive projects (except base and initial project)
9	Icon indicates that the project is pinned.
10	At this point in time, the project is automatically deleted unless it is activated beforehand. The time can be set for each project in WorkVisual by means of **Time for activation:**. The default setting is that the project is never deleted. As soon as the project on the robot controller is, or has been, active, this setting no longer has any effect: the project is not automatically deleted.

With all copying operations, a window opens in which a name and a description can be entered for the copy.

Buttons

The following buttons are available:

Button	Description
Activate	Activates the selected project. If the selected project is pinned: creates a copy of the selected project. (A pinned project cannot be activated itself, only a copy of it.) The user can then decide whether to activate this copy or whether the current project should remain active. Precondition: user group “Expert” If the project to be activated contains changes in the safety settings: user group “Safety maintenance”
Pin	Pins the project. (>>> *7.11.1 "Pinning a project on the robot controller" Page 311*) Only available if an unpinned project is selected. Required user rights: function group **General configuration**
Unpin	Unpins the project. Only available if a pinned project is selected. Required user rights: function group **General configuration**
Copy	Copies the selected project.
Delete	Deletes the selected project. Only available if a non-activated, unpinned project is selected. Required user rights: function group **General configuration**
Edit	Opens a window in which the name and/or description of the selected project can be changed.
Refresh	Refreshes the project list. This enables, for example, projects to be displayed which have been transferred to the robot controller since the display was opened.

7.11.3.2 Restoration points tab

Special projects

Special projects area:

The area is the same as on the **Projects** tab.

Available projects

Available projects area:

A list of recently active projects is displayed here. Date and time indicate when the project was saved, i.e. the time at which the next project was activated.

The user can activate and reactivate a project. The project, including its options, is reset to active. The previously active project is automatically saved, including its options, and moved to the list.

- A maximum of 5 projects are displayed. If a further project is activated, the project in the list that was activated least recently is deleted.

- If a project has been activated repeatedly, it is displayed several times in the list.

Particular advantage: If a project is activated that has the same name as the project that is already active, the latter is not overwritten, but saved in the list. It is thus not lost and can be reactivated subsequently if required.

Buttons

The following buttons are available:

Button	Description
Restore	Activates the selected project. Precondition: user group “Expert” If the project to be activated contains changes in the safety settings: user group “Safety maintenance”
Delete	Deletes the selected project.
Refresh	Refreshes the project list. This enables, for example, projects to be displayed which have been transferred to the robot controller since the display was opened.

7.12 Backup Manager

7.12.1 Overview of Backup Manager

Overview

The Backup Manager makes it possible to back up and restore projects, option packages and RDC data.

The default settings for the Backup Manager are defined in the **Backup configuration** window. The settings can be changed.

Start types

There are several ways of starting a backup or restoration:

	Backup	Restoration
Manual	Yes	Yes
Automatic, in intervals	Yes	No
Via inputs KSS: only in AUT or AUT EXT VSS: only in EXT	Yes	Yes (only projects and option packages, no RDC data)

When a backup/restoration is running, no further backup/restoration can be started. However, the start types are not mutually exclusive. If, for example, an automatic backup is configured, a manual backup can still be executed.

	Execution
Manual	(>>> *7.12.2 "Backing up projects, option packages and RDC data manually" Page 317*) (>>> *7.12.3 "Manually restoring projects and option packages" Page 318*) (>>> *7.12.4 "Restoring RDC data manually" Page 320*)
Automatic, in intervals	If backups are to be started automatically, this must be configured in the **Backup configuration** window.
Via inputs	If backups/restorations are to be started via inputs, this must be configured in the **Backup configuration** window.

7.12.2 Backing up projects, option packages and RDC data manually

Projects

The following projects are backed up by default:

- Active project
- Initial project
- Base project

Option packages

Option packages will be backed up under the following conditions:

- The option package has a KOP file.
- The option package was originally added to the project in WorkVisual. The project is now active on the robot controller.

  Or:

  The option package has been installed in the active project via **Start-up** > **Additional software**. The option package was available during installation as a single KOP file (not as a directory structure!).

RDC data

Every time a backup is made, a file with the name *[Robot_serial_number]*.RDC is created. It contains the CAL, MAM and PID files. Not all files are present in all cases (dependent on the robot).

Menu items

There are 2 menu items available for backup. They differ with regard to the target paths.

- **Back up**

  Backup is carried out to the paths stored in the following boxes in the **Backup configuration** window: **Target path for project backup** and **Target path for KOP backup**.
- **Save as...**

  A path can be set here. It only applies to this backup. The option packages are also backed up to this path by default. If necessary, however, a separate path can be set for the option packages.

If the paths do not yet exist, they are automatically created during the backup operation.

Precondition

- User rights: Function group **Archive with unknown destination**

Procedure

Back up:

- In the main menu, select **File** > **Backup Manager** > **Back up**.

Save as...:

1. In the main menu, select **File** > **Backup Manager** > **Save as...**.
2. Enter the target in the **Target path for project backup** box.
3. If necessary, enter a separate path for the option packages.

   To do so, activate the check box **Divert path for option packages** and enter the desired path in the **Target path for KOP backup** box.
4. Start the backup with **Save**.

The backup is carried out. The robot controller displays messages when the backup has been successfully completed. It generates one message per project or option package and one message relating to the RDC data.

However, option packages will not be backed up if the same package version already exists in the target directory.

7.12.3 Manually restoring projects and option packages

Projects

The following projects are restored by default:

- Active project (i.e. the project which was active during the backup)
- Initial project
- Base project

After manual restoration of the project, the robot controller reactivates the previously active project if certain preconditions are met. If not, the project remains inactive. This can be activated by the user at the desired point in time.

After restoration of the project via inputs, the previously active project also remains inactive.

Option packages

The option packages for which there are KOP files in the source directory are not necessarily all restored. An option package will be restored under the following conditions:

- At the time of backup, the option package was present in the active project.
- At the time of restoration, this version of the option package is not installed on the robot controller.

Following restoration, some option packages automatically install themselves when the corresponding project is activated.

Option packages that do not automatically install themselves are available for installation under **Start-up** > **Additional software**. A message on the robot controller indicates that the option package must still be installed.

Menu items

There are 2 menu items available for restoration. They differ with regard to the paths that are accessed.

- **Restore**

  The paths stored in the following boxes in the **Backup configuration** window are accessed: **Target path for project backup** and **Target path for KOP backup**.
- **Restore from ...**

  A path can be set here. It only applies to this restoration. If necessary, a separate path can be set for the option packages.

Precondition

- User rights: function group **Restore**

  But at least the user group “Expert”

> If a safety option is installed on the robot controller, e.g. KUKA.SafeOperation, different preconditions may apply. Information can be found in the documentation of the safety options.

Procedure

Restore:

- In the main menu, select **File** > **Backup Manager** > **Restore** > **Projects and options**.

Restore from ... :

1. In the main menu, select **File** > **Backup Manager** > **Restore from ...** > **Projects and options**.
2. Enter the source path in the **Source path of the project to be restored** box.
3. If necessary, enter a separate path for the option packages.

   To do so, activate the check box **Divert path for option packages** and enter the desired path in the **Source path for KOP** box.
4. Start restoration with **Restore**.

   The robot controller displays a message for each project and option package when the restoration has been successfully completed.
5. Project activation:
   - Operating mode T1/T2:

     The robot controller displays the following request for confirmation: *Do you want to activate the project […]?* In addition, a message is displayed as to whether the activation would overwrite a project and, if so, which one.

     If no relevant project will be overwritten: Confirm with **Yes** within 30 minutes.
   - Operating mode AUT/AUT EXT:

     The robot controller starts to activate the project without a request for confirmation.
6. An overview is displayed of the changes which will be made in comparison to the project that is still active on the robot controller. The check box **Details** can be used to display details about the changes.

WARNING

If changes are listed in the overview under the heading **Safety-relevant communication parameters**, this means that the behavior of the Emergency Stop and “Operator safety” signal may have changed compared with the previous project.
After activation of the project, the Emergency Stop and the “Operator safety” signal must be checked for safe functioning. If the project is activated on several robot controllers, this check must be carried out for every robot controller. Failure to carry out this check may result in death, injuries or damage to property.

7. The overview displays the request for confirmation *Do you want to continue?*. Confirm with **Yes**. The project is activated on the robot controller.

WARNING

After activation of a project on the robot controller, the safety configuration must be checked there! If this is not done, the robot will possibly be operated with incorrect data. Death, injuries or damage to property may result.
(>>> *6.6 "Checking the safety configuration of the robot controller" Page 215*)

WARNING

If the activation of a project fails, an error message is displayed. In this case, one of the following measures must be carried out:

- Either: Activate a project again (the same one or a different one).
- Or: Reboot the robot controller with a cold restart.

7.12.4 Restoring RDC data manually

Menu items

There are 2 menu items available for restoration. They differ with regard to the path that is accessed.

- **Restore**
  The path stored under **Target path for project backup** in the **Backup configuration** window is accessed.
- **Restore from ...**
  A path can be set here. It only applies to this restoration.

Precondition

- User rights: Function group **Restoration of critical data**

Procedure

Restore:

- In the main menu, select **File** > **Backup Manager** > **Restore** > **RDC data**.

Restore from ... :

1. In the main menu, select **File** > **Backup Manager** > **Restore from ...** > **RDC data**.
2. Select the source path.

3. A window opens. Activate the check boxes next to the data that are to be restored:
   - **PID file**, **MAM file** and/or **CAL file**

   If an entry is grayed out, this means that this file is not available in the backup.
4. Confirm the selection with **Restore**.

Once the restoration has been completed, the following message is displayed: *RDC data successfully restored from {0}.*

7.12.5 Configuring Backup Manager

Precondition

- User rights: Function group **General configuration**
- For changes to the **Signal interface** tab additionally:
  T1 or T2 mode

Procedure

1. In the main menu, select **File** > **Backup Manager** > **Backup configuration**.
   The **Backup configuration** window is opened.
2. Carry out the desired changes in the tabs.
   - **Backup configuration** contains the general settings.
     Automatic saving can also be configured here if required.
   - I/O control can also be configured under **Signal interface** if required.
3. Close the window.
4. Respond to the request for confirmation asking whether the changes should be saved by pressing **Yes**.

7.12.5.1 “Backup configuration” tab

Backup configuration

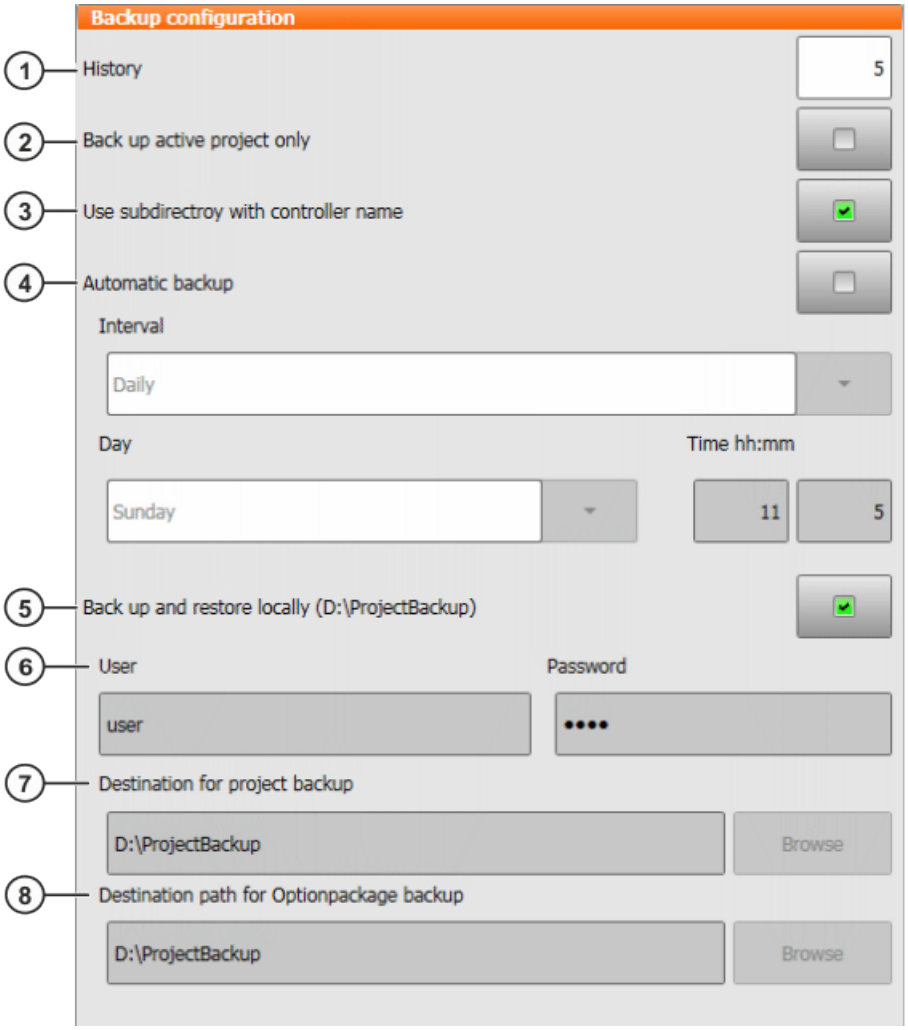


Fig. 7-16: “Backup configuration” tab

Item	Description
1	Maximum number of subfolders for older backups A backup does not overwrite existing backup files. Instead, a subfolder is automatically created and the files are transferred there. When the maximum number of subfolders has been reached, the oldest subfolder will be deleted during the next backup and a new one will be created. • **0 … 50** Default: 5
2	• **Activated**: The active project is backed up during backup. During restoration, only the project which was active during the backup will be restored. • **Deactivated** (default): The following projects are backed up during the backup: Active project, base project, initial project These projects are restored during restoration.

Item	Description
3	• **Active** (default): During backup, the robot controller stores the projects and RDC data in the path [**Target path for project backup**]\[controller name]\. The robot controller accesses this path during restoration. **Note:** Select this setting if more than one robot controller uses the same target path. • **Inactive**: During backup, the robot controller stores the projects and RDC data in the path [**Target path for project backup**]\. The robot controller accesses this path during restoration. **Note:** Option packages are always stored under [**Target path for KOP backup**]\OptionPackages\ irrespective of this setting.
4	• **Active**: The robot controller automatically carries out backups. The following parameters determine the point in time: **Interval**, **Day**, **Time hh:mm** **Note:** After an automatic backup, the robot controller does not display a message of successful completion. If the robot controller was switched off at the configured time, it carries out a backup as soon as it is switched on again. It only carries out one backup, even if the time was missed more than once. • **Deactivated** (default): No automatic backup.
5	• **Active** (default): The target path for backups and the source path for restorations is D:\ProjectBackup. • **Inactive**: The following parameters determine the target path for backups and the source path for restorations: **Target path for project backup**, **Target path for KOP backup**
The following parameters can only be edited if the option **Back up and restore locally (D:\ProjectBackup)** is **deactivated**.	
6	If during backup and/or restoration the robot controller must access the network and an authentication is required, the data saved here are used. If no authentication is required, the data have no effect. • **User**: User name Default: user • **Password**: Password. On entry, only the number of characters is displayed and not the characters themselves. Default: kuka
7	For projects and RDC data: Target path for backups and source path for restorations It is possible to navigate to the desired directory or to enter it directly. In the latter case, the path is automatically created if it does not already exist.
8	For KOP files: Target path for backups and source path for restorations It is possible to navigate to the desired directory or to enter it directly. In the latter case, the path is automatically created if it does not already exist.

7.12.5.2 “Signal interface” tab

When a restoration is started via inputs, the robot controller – in contrast to manual restoration – does not activate the previously active project. This can be activated by the user at the desired point in time.

Signal interface

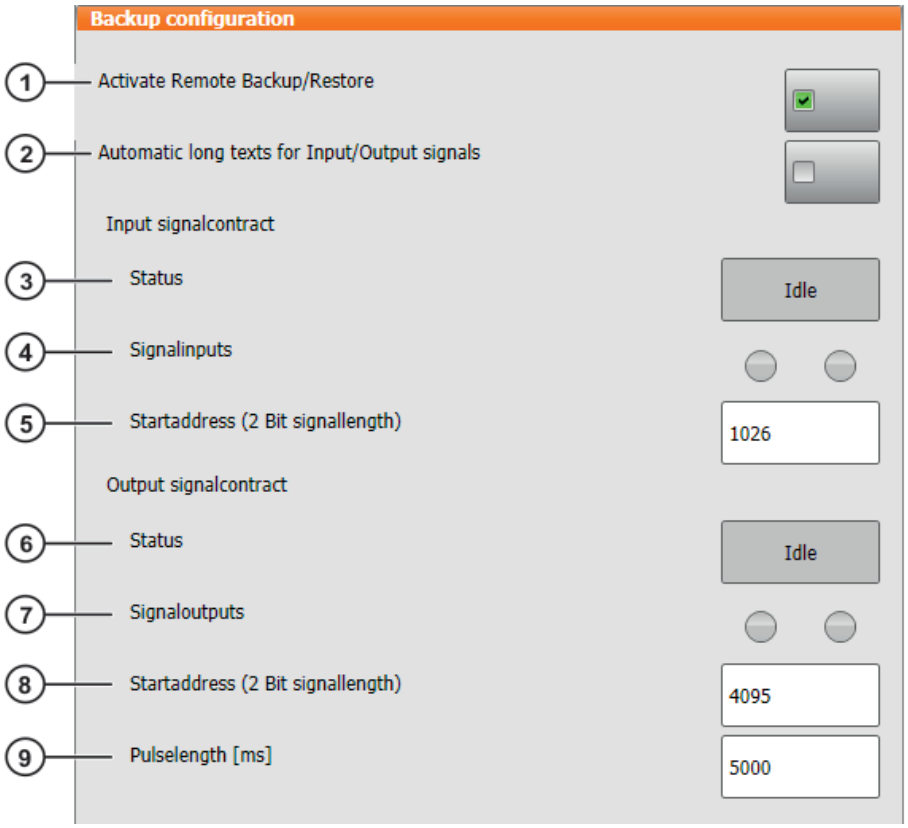


Fig. 7-17: “Signal interface” tab

Item	Description
1	• **Activated**: Backups and restorations can be started via the input signal configured under **Start address**. • **Deactivated** (default): Start is not possible via input signal.
The following parameters can only be edited if the option **Activate Remote Backup/Restore** is **activated**.	
2	• **Activated**: The signals defined under **Start address** are assigned the following long text names: **BM input signal**, **BM output signal** If the signals already have long text names, these are not overwritten. • **Deactivated** (default): No long text names If **Activated** was previously selected, **Deactivated** removes the long text names.
3	Displays when the input signal starts a backup or restoration. (Display during the length of a signal) (>>> *"Input signals" Page 325*)

Item	Description
4	• Left-hand lamp: Status of the input **Start address**. • Right-hand lamp: Status of the input **Start address** + 1. Status (cannot be edited): Green = 1; Gray = 0
5	Input signal consisting of the following inputs: **Start address** and **Start address** + 1. Default: 1 026
6	Displays the action being executed or the result. (>>> *"Output signals" Page 325*)
7	• Left-hand lamp: Status of the output **Start address**. • Right-hand lamp: Status of the output **Start address** + 1. Status (cannot be edited): Green = 1; Gray = 0
8	Output signal consisting of the following outputs: **Start address** and **Start address** + 1. Default: 4 095
9	Duration for which output signal 01 or 11 is present before the robot controller sets 00 again Unit: ms Default: 5,000 ms

Input signals

Signal	→ Created actions / display in the Status box
00	No action / **Idle**
01	Starts a backup / **Backup**
10	Starts a restoration / **Restore**
11	Undefined state

A continuously active input signal does not cause the action to repeat.

When a backup/restoration is running, no further backup/restoration can be started.

Output signals

The signals 01 and 11 are present during the time configured in **Pulse duration [ms]**, after which point the signal switches to 00.

Action / Result	→ Created signal / display in the Status box
No action running	00 / **Idle**
Backup running	10 / **Backup**
Restoration running	10 / **Restore**
Backup successful	01 / **BackupSuccess**
Restoration successful	01 / **RestoreSuccess**
Backup unsuccessful	11 / **BackupError**
Restoration unsuc-cessful	11 / **RestoreError**

8 Program execution

8.1 Selecting the program run mode

Procedure

1. Touch the status indicator **Program run mode**. The **Program run mode** window opens.

Fig. 8-1: Program run mode status indicator

2. Select the desired program run mode.
   The window closes and the selected program run mode is applied.

8.2 Program run modes

Designation	Status indicator	Description
Go #GO		The program is executed through to the end without stopping. Required user rights: Function group **Program execution settings**
Motion #MSTEP		The program is executed with a stop at each point, including auxiliary points and the points of a spline segment. The Start key must be pressed again for each point. The program is executed without advance processing. Required user rights: Function group **Program execution settings**
Single Step #ISTEP		The program is executed with a stop after each program line. The motion is also stopped after program lines that cannot be seen and after blank lines. The Start key must be pressed again for each line. The program is executed without advance processing. Required user rights: Function group **Critical jog settings**

Designation	Status indicator	Description
Backwards #BSTEP		This program run mode is automatically selected if the Start backwards key is pressed. It is not possible to select a different mode. This mode works in the same way as **Motion**, but with the following exception: CIRC motions are executed backwards in the same way as they were last executed forwards, i.e. if the forward motion was not stopped at the auxiliary point, the backward motion will not be stopped there either. This exception does not apply in the case of SCIRC motions. Here, the backward motion is always stopped at the auxiliary point.

The following additional program run modes are available for systems integrators.

These program run modes can only be selected via the variable correction function. System variable for the program run mode: $PRO_MODE.

Designation	Status indicator	Description
Program Step #PSTEP		The program is executed step by step without advance processing. Subprograms are executed completely.
Continuous Step #CSTEP		Approximate positioning points are executed with advance processing, i.e. they are approximated. Exact positioning points are executed without advance processing and with a stop after the motion instruction.

8.3 Advance run

The advance run is the maximum number of motion blocks that the robot controller calculates and plans in advance during program execution. The actual number is dependent on the capacity of the computer.

The advance run refers to the current position of the block pointer. It is set via the system variable $ADVANCE:

- Default value: 3
- Maximum value: 5

The advance run is required, for example, in order to be able to calculate approximate positioning motions. If $ADVANCE = 0 is set, approximate positioning is not possible.

Certain statements trigger an advance run stop. These include statements that influence the periphery, e.g. OUT statements.

8.4 Block pointer

Overview

During program execution, the block pointer indicates various items of information:

- Which motion the robot is currently executing or has completed
- Whether an auxiliary point or end point is currently being approached
- The direction in which the robot is executing the program

Pointer	Direction	Description
	Forwards	The end point is being approached.
	Backwards	
	Forwards	The end point has been reached with exact positioning.
	Backwards	
	Forwards	The auxiliary point is being approached.
	Backwards	
	Forwards	The auxiliary point has been reached with exact positioning.
	Backwards	

Examples for forward motion

```
5  PTP P3 Vel=100 % PDAT1 Tool[1] Base[0]
6  PTP P4 Vel=100 % PDAT2 Tool[1] Base[0]
7  PTP P5 Vel=100 % PDAT3 Tool[1] Base[0]
```

Fig. 8-2: The robot is moving from P3 to P4

```
5  PTP P3 Vel=100 % PDAT1 Tool[1] Base[0]
6  PTP P4 Vel=100 % PDAT2 Tool[1] Base[0]
7  PTP P5 Vel=100 % PDAT3 Tool[1] Base[0]
```

Fig. 8-3: The robot has reached P4 with exact positioning

```
6  PTP P5 Vel=100 % PDAT3 Tool[1] Base[0]
7  CIRC P6 P7 Vel=2 m/s CPDAT1 Tool[1] Base[0]
8  PTP P8 Vel=100 % PDAT16 Tool[1] Base[0]
```

Fig. 8-4: The robot is moving from P5 to auxiliary point P6

```
6  PTP P5 Vel=100 % PDAT3 Tool[1] Base[0]
7  CIRC P6 P7 Vel=2 m/s CPDAT1 Tool[1] Base[0]
8  PTP P8 Vel=100 % PDAT16 Tool[1] Base[0]
```

Fig. 8-5: The robot has reached auxiliary point P6 with exact positioning

```
6  PTP P5 Vel=100 % PDAT3 Tool[1] Base[0]
7  CIRC P6 P7 Vel=2 m/s CPDAT1 Tool[1] Base[0]
8  PTP P8 Vel=100 % PDAT16 Tool[1] Base[0]
```

Fig. 8-6: The robot is moving from auxiliary point P6 to P7

```
6  PTP P5 Vel=100 % PDAT3 Tool[1] Base[0]
7  CIRC P6 P7 Vel=2 m/s CPDAT1 Tool[1] Base[0]
8  PTP P8 Vel=100 % PDAT16 Tool[1] Base[0]
```

Fig. 8-7: The robot has reached P7 with exact positioning

Examples for backward motion

```
6  PTP P5 Vel=100 % PDAT3 Tool[1] Base[0]
7  CIRC P6 P7 Vel=2 m/s CPDAT1 Tool[1] Base[0]
8  PTP P8 Vel=100 % PDAT16 Tool[1] Base[0]
```

Fig. 8-8: The robot is moving from P8 to P7

```
6  PTP P5 Vel=100 % PDAT3 Tool[1] Base[0]
7  CIRC P6 P7 Vel=2 m/s CPDAT1 Tool[1] Base[0]
8  PTP P8 Vel=100 % PDAT16 Tool[1] Base[0]
```

Fig. 8-9: The robot has reached P7 with exact positioning

```
6  PTP P5 Vel=100 % PDAT3 Tool[1] Base[0]

7 CIRC P6 P7 Vel=2 m/s CPDAT1 Tool[1] Base[0]

8  PTP P8 Vel=100 % PDAT16 Tool[1] Base[0]
```

Fig. 8-10: The robot is moving from P7 to auxiliary point P6

```
6  PTP P5 Vel=100 % PDAT3 Tool[1] Base[0]

7 CIRC P6 P7 Vel=2 m/s CPDAT1 Tool[1] Base[0]

8  PTP P8 Vel=100 % PDAT16 Tool[1] Base[0]
```

Fig. 8-11: The robot has reached auxiliary point P6 with exact positioning

Double upward/downward arrow

If the program window shows a section in which the block pointer is not currently located, a double arrow indicates the direction in which it is to be found.

```
7  PTP P6 Vel=100 % PDAT4 Tool[1] Base[0]

8  PTP P7 Vel=100 % PDAT5 Tool[1] Base[0]
```

Fig. 8-12: The block pointer is located higher up in the program

```
14  PTP P13 Vel=100 % PDAT11 Tool[1] Base[0]

15  PTP P14 Vel=100 % PDAT12 Tool[1] Base[0]
```

Fig. 8-13: The block pointer is located lower down in the program

8.5 Setting the program override

Description

Program override is the velocity of the robot during program execution. The program override is specified as a percentage of the programmed velocity.

In T1 mode, the maximum velocity is 250 mm/s, irrespective of the value that is set.

Precondition

- User rights: Function group **Program execution settings**

Procedure

1. Touch the status indicator **Overrides**. The **Overrides** window opens.

Fig. 8-14: Overrides status indicator

2. Set the desired program override. It can be set using either the plus/minus keys or by means of the slider.
   - Plus/minus keys: The override can be adjusted in predefined steps.
   - Slider: The override can be adjusted in 1% steps.
3. Touch the status indicator **Overrides** again. (Or touch the area outside the window.)

   The window closes and the selected override value is applied.

Alternative procedure

Alternatively, the override can be set using the plus/minus key on the lower right-hand side of the smartPAD.

The value can be set to 100%, 75%, 50%, 30%, 10%, 5%, 3%, 1%.

8.6 Robot interpreter status indicator

Icon	Color	Description
	Gray	No program is selected.
	Yellow	The block pointer is situated on the first line of the selected program.
	Green	The program is selected and is being executed.
	Red	The selected and started program has been stopped.
	Black	The block pointer is situated at the end of the selected program.

8.7 Starting a program forwards (manual)

Precondition

- A program is selected.
- Operating mode T1 or T2

Procedure

1. Select the program run mode.
2. Hold the enabling switch down and wait until the status bar indicates “Drives ready”:

Fig. 8-15

3. Carry out a BCO run: Press Start key and hold it down until the message "*Programmed path reached (BCO)*" is displayed in the message window. The robot stops.

CAUTION
The BCO run is executed as a LIN or PTP motion from the actual position to the target position. The velocity is automatically reduced. The path of the motion cannot be predicted reliably. Observe the motion during the BCO run so that the robot can be stopped in time if a collision becomes imminent.

4. Press Start key and hold it down.

   The program is executed with or without stops, depending on the program run mode.

To stop a program that has been started manually, release the Start key.

8.8 Starting a program forwards (automatic)

Precondition

- A program is selected.
- Operating mode Automatic (not Automatic External)

Procedure

1. Select the program run mode **Go**.
2. Switch on the drives.
3. Carry out a BCO run:

   Press Start key and hold it down until the message "Programmed path reached (BCO)" is displayed in the message window. The robot stops.

CAUTION
The BCO run is executed as a LIN or PTP motion from the actual position to the target position. The velocity is automatically reduced. The path of the motion cannot be predicted reliably. Observe the motion during the BCO run so that the robot can be stopped in time if a collision becomes imminent.

4. Press the Start key. The program is executed.

To stop a program that has been started in Automatic mode, press the STOP key.

8.9 BCO run

BCO stands for "block coincidence". The robot controller performs a so-called "BCO run" in the following cases:

- If the Start key is pressed after selecting or resetting the program
- If the Start key is pressed after selecting a motion block using the **Block selection** button

- If program execution has been interrupted and the program modified, and the program is then resumed.

Before the BCO run, there is no guarantee that the TCP is on the path. The BCO run is required in order to move the TCP onto the path, generally to a programmed point. Following selection of a program, e.g. the BCO run goes to the first programmed point in the program (generally the HOME position).

When the BCO run has been successfully completed, the robot controller indicates this by means of a corresponding message. Only then can it plan the (further) path. The user can now start the actual program by means of a renewed start action.

If the robot is already at, or very close to, a defined position on the path before the BCO run, the BCO run is very short. The robot moves very little or not at all. The user can tell that a BCO run has been carried out because the robot controller generates a corresponding message.

CAUTION
The BCO run is executed as a LIN or PTP motion. The velocity is automatically reduced. The path of the motion cannot be predicted reliably. Observe the motion during the BCO run so that the robot can be stopped in time if a collision becomes imminent.

8.10 Carrying out a block selection

Description

A program can be started at any point by means of a block selection.

Precondition

- A program is selected.
- Operating mode T1 or T2

Procedure

1. Select the program run mode.
2. Select the motion block at which the program is to be started.
3. Press **Block selection**. The block pointer indicates the motion block.
4. Hold the enabling switch down and wait until the status bar indicates “Drives ready”:

5. Carry out a BCO run: Press the Start key and hold it down until the message “*Programmed path reached (BCO)*” is displayed in the message window. The robot stops.

CAUTION
The BCO run is executed as a LIN or PTP motion from the actual position to the target position. The velocity is automatically reduced. The path of the motion cannot be predicted reliably. Observe the motion during the BCO run so that the robot can be stopped in time if a collision becomes imminent.

6. The program can now be started manually or automatically. It is not necessary to carry out a BCO run again.

8.11 Resetting a program

Description

In order to restart an interrupted program from the beginning, it must be reset. This returns the program to the initial state.

Precondition

- Program is selected.

Procedure

- Select the menu sequence **Edit** > **Reset program**.

Alternative procedure

- In the status bar, touch the **Robot interpreter** status indicator. A window opens.

  Select **Reset program**.

8.12 Starting Automatic External mode

NOTICE

There is no BCO run in Automatic External mode. This means that the robot moves to the first programmed position after the start at the programmed (not reduced) velocity and does not stop there.

Precondition

- Operating mode T1 or T2
- Inputs/outputs for Automatic External are configured.
- The program CELL.SRC is configured.

Procedure

1. Select the program CELL.SRC in the Navigator. (This program is located in the folder “R1”.)
2. Set program override to 100%. (This is the recommended setting. A different value can be set if required.)
3. Carry out a BCO run:

   Hold down the enabling switch. Then press the Start key and hold it down until the message “Programmed path reached (BCO)” is displayed in the message window.

   CAUTION

   The BCO run is executed as a LIN or PTP motion from the actual position to the target position. The velocity is automatically reduced. The path of the motion cannot be predicted reliably. Observe the motion during the BCO run so that the robot can be stopped in time if a collision becomes imminent.

4. Select “Automatic External” mode.
5. Start the program from a higher-level controller (PLC).

To stop a program that has been started in Automatic mode, press the STOP key.

8.13 Backward motion using the Start backwards key

In addition to backward motion using the “Start backwards” key, there is another option for backward motion. Information about it can be found here:
(>>> *4.16.10 "Backward motion using the jog keys" Page 92*)
An overview of the most important differences can be found here:
(>>> *8.13.4 "Comparison of “Start backwards”/backwards using the jog keys" Page 341*)

8.13.1 Executing motions backwards (using the “Start backwards” key)

Description

Backward motion via the “Start backwards” key is often used if a sequence of motions is to be optimized and individual points are to be re-taught for this purpose. The user executes the path backwards until the point that is to be corrected has been reached. Once the point has been re-taught, backward motion is continued if required in order to correct further points.

The program run mode #BSTEP is automatically applied for backward motion.

Approximate positioning and weaving are not possible during backward motion. If approximate positioning or weaving were carried out for points during forward execution, the backward path will thus differ from the forward path. It is thus possible that the robot may have to perform a BCO run after starting backward motion, even though it did not leave the path during forward motion.

CAUTION

The BCO run is executed as a LIN or PTP motion. The velocity is automatically reduced. The path of the motion cannot be predicted reliably. Observe the motion during the BCO run so that the robot can be stopped in time if a collision becomes imminent.

Precondition

- A program is selected.
- The motions that are to be executed backwards have been executed forwards.
- T1 or T2 mode

Procedure

1. Hold the enabling switch down and wait until the status bar indicates “Drives ready”:

2. Press and hold down the Start backwards key.
   - If the robot is already on the backward path, it now moves backwards.
   - If the robot is not on the backward path, it now moves to it. When “*Programmed path reached (BCO)*” is displayed in the message window, it has reached the path. The robot stops.

     Press the Start backwards key again. The robot now moves backwards.

3. Press the Start backwards key again for each motion block.

8.13.2 Functional principle and characteristics of backward motion

Functional principle

During forward motion, the robot controller saves the executed motions in a ring buffer. During backward motion, the motions are executed on the basis of the saved information.

No backward motion possible once the buffer has been deleted:

The contents of the buffer are deleted in the following cases. Backward motion is not possible again until motions have been executed in the forward direction again.

- Program is reset.
- Program is deselected.
- Lines are inserted into the program or deleted.
- KRL instruction RESUME
- Block selection to a motion other than the current one.

  What is possible without restriction, however, is a block selection to any segment point within the current spline block. This counts as block selection to the current motion, as the robot controller plans and executes the spline block as one motion.

The robot controller deletes the buffer without generating a corresponding message.

Properties

- Backward motion is only possible in modes T1 and T2.
- Only motions are executed during backward motion, and no control structures or control instructions.
- Outputs and flags are not recorded during forward motion. For this reason, their previous states are not restored during backward motion.
- The velocity is the same as for forward motion.

  In T2, it is possible that monitoring functions may be triggered during backward motion that are not triggered during forward motion. In this case, the program override must be reduced.

Backward motion can be deactivated. Further configuration options are also available.

(>>> *6.18 "Configuring backward motion" Page 237*)

`DELETE_BACKWARD_BUFFER()` can be used to prevent backward motion for specific motions.

(>>> *11.15.1 "DELETE_BACKWARD_BUFFER(): preventing backward start" Page 553*)

Torque/force mode, VectorMove

The following applies to motions with torque or force mode or VectorMove:

- Backward motion is possible for conventional motions, but force/torque mode or VectorMove is automatically deactivated.

- Spline motions cannot be executed backwards.

NOTICE

In the case of more complex applications with torque mode (e.g. press ejection) and/or VectorMove, backward motion is generally not recommended, as the underlying processes are not usually reversible.
In such cases, it is advisable to prevent backward motion using DELETE_BACKWARD_BUFFER(). Damage to property may otherwise result.

8.13.2.1 Response in the case of subprograms

- Motions executed forwards in an interrupt program are not recorded. They cannot, therefore, be executed backwards.
- If a subprogram has been completely executed during forward motion, it cannot be executed with backward motion.
- If the forward motion was stopped in a subprogram, the response depends on the position of the advance run pointer:

Position of the advance run pointer	Response
Advance run pointer is in the subprogram.	Backward motion is possible.
Advance run pointer has already left the subprogram.	Backward motion is not possible. Prevention: Trigger an advance run stop before the END of the subprogram, e.g. with WAIT SEC 0. However, it is then no longer possible to carry out approximate positioning at this point. Or set $ADVANCE to "1". This does not always prevent the error message, but it reduces the probability. Approximate positioning is still possible.

8.13.2.2 Approximate positioning response

Description

Approximate positioning is not possible during backward motion. If approximate positioning was carried out for points during forward execution, the backward path will thus differ from the forward path. It is thus possible that the robot may have to perform a BCO run for the backward path after starting backward motion, even though it did not leave the path during forward motion.

Example 1

Backward start outside an approximate positioning range:

The Start backwards key is pressed while the robot is on the path, but not in an approximate positioning range. The robot now moves backwards on the path to the end point of the previous motion.

P_{BACK} = position of the robot at the moment at which the Start backwards key is pressed

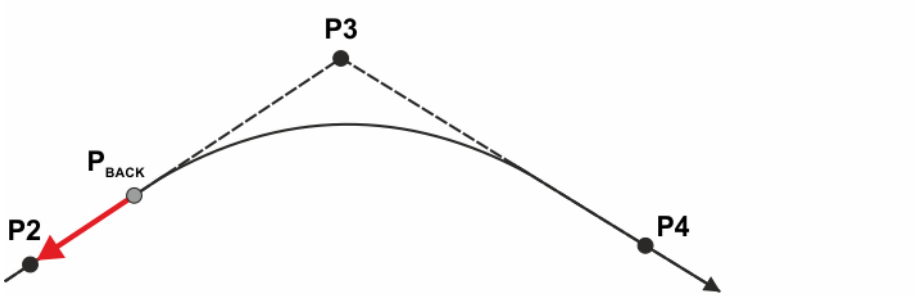


Fig. 8-16: Case 1: Backward start outside an approximate positioning range

If the end point of the previous motion is approximated, it is nonetheless addressed with exact positioning.

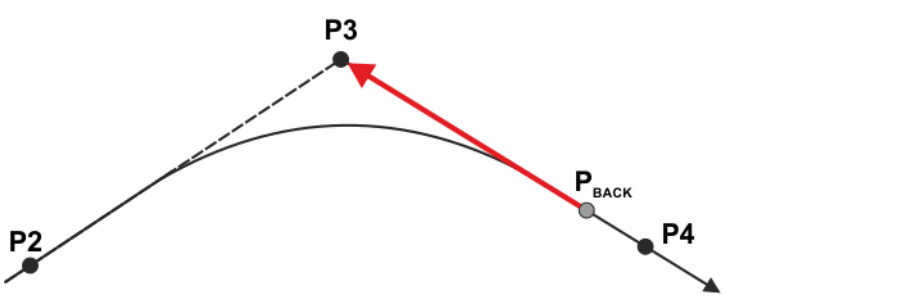


Fig. 8-17: Case 2: Backward start outside an approximate positioning range

Example 2

Backward start in the approximate positioning range:

The Start backwards key is pressed while the robot is in an approximate positioning range. The robot now performs a BCO run to the start of the approximate positioning range and stops there. If the Start backwards key is now pressed again, the actual backward motion begins, i.e. the robot moves backwards along the path to the end point of the previous motion.

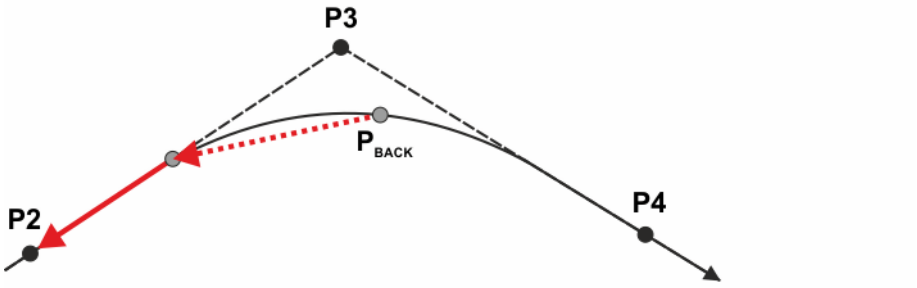


Fig. 8-18: Backward start in the approximate positioning range

8.13.2.3 Response in the case of weave motions

Description

Weaving is not possible during backward motion. If weaving was carried out during forward execution, the backward path will thus differ from the forward path. The robot must therefore perform a BCO run for the backward path after starting backward motion, even though it did not leave the path during forward motion.

Example

Backward start on weave path:

The Start backwards key is pressed while the robot is weaving. The robot now performs a BCO run to the taught path and stops there. If the Start backwards key is now pressed again, the actual backward motion begins, i.e. the robot moves backwards along the path to the end point of the previous motion.

P_{BACK} = position of the robot at the moment at which the Start backwards key is pressed

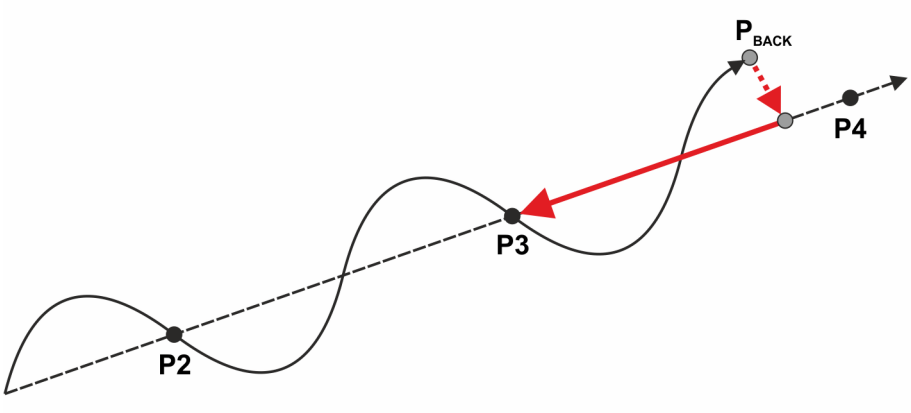


Fig. 8-19: Backward start on weave path

8.13.2.4 Switching from backwards to forwards

Precondition

It is only possible to resume forward motion following backward motion if the following preconditions are met:

- Block selection is possible to the program line on which the backward block pointer is currently located.
- If the first motion to be executed forwards again is a conventional motion, it must be completely programmed.

  It is thus not possible, for example, to switch from backward motion to forward motion if the first motion is a PTP_REL motion.

  With few exceptions, this restriction does not apply in the case of spline motions.

Response

When the Start forward key is pressed for the first time following backward motion, the response is as follows:

- If BCO exists, the program run mode most recently used in the forward direction is automatically restored and the robot moves forwards on the path.
- If BCO does not exist, a BCO run is carried out. The program run mode meanwhile remains set to #BSTEP. After the BCO run, the robot stops. The Start forwards key must now be pressed again. The program run mode most recently used in the forward direction is automatically restored and the robot now moves forwards on the path.

If the switch from backwards to forwards occurs in a control structure, the robot first moves forwards to the end of the control structure. It then stops with the message *Control structure next block {Block number}*. The block number specifies the first block after the control structure.

8.13.3 System variables with changed meaning

The meaning of certain system variables relating to backward motion has changed with effect from V8.3.6. In some cases, they no longer have any effect, but still exist for reasons of compatibility with older versions.

The system variables that start with “$VW_…” exist not only in the VSS, but also in the KSS.

Designation	Meaning in V8.3.6 and higher
$BWD_INFO	Contains the current configuration for backward motion as a bitmap. Numerous technology packages read this variable. The variable can only be read.
$BWDSTART	This variable only still exists for reasons of compatibility. It can be set to TRUE or FALSE, but this has no effect.
$VW_BACKWARD	Corresponds to the attribute ENABLE of the configuration element BACKWARD_STEP. The variable can only be read.
$VW_CYCFLAG	Always supplies the value 0. This variable only still exists for reasons of compatibility and can only be read.
$VW_MOVEMENT	Corresponds to the attribute MOVEMENTS of the configuration element BACKWARD_STEP. The variable can only be read.
$LOAD_BWINI	Always supplies the value FALSE. This variable only still exists for reasons of compatibility and can only be read.

8.13.4 Comparison of “Start backwards”/backwards using the jog keys

The table shows the most important differences between backward motion with the “Start backwards” key and backward motion with the jog keys.

Other backward motion functionalities, e.g. backward motion as part of fault strategies in technology packages, are not taken into consideration here.

Using “Start backwards”	Using the jog keys
Program motions can be executed backwards.	Virtually all types of motions can be executed backwards.
The path may differ from the forward path.	The path is the same as the forward path.
Approximate positioning and weaving are not carried out during backward motion.	
The original motions can be executed backwards individually.	The path is a continuum. It can be stopped at any point, but the original motions cannot be executed backwards individually (motion by motion).
Program run mode = #BSTEP, i.e. stop after each program line	Fewer or at most the same number of exact positioning points as the forward path

8.14 Collision detection

8.14.1 Overview of collision detection

Functionality

If the robot collides with an object, the robot controller increases the axis torques in order to overcome the resistance. This can result in damage to the robot, tool or other objects.

Collision detection reduces the risk of such damage. It monitors the axis torques. If these deviate from a specified tolerance range, the following reactions are triggered:

- The robot stops with a STOP 2.
- Message *Ackn.: Collision detection axis {Axis number}*
- The signal $COLL_ALARM is set to TRUE.
- The robot controller calls the program CollDetect_UserAction.

  The program is located in the folder R1\Program. It does not contain any instructions as standard. If required, the user can program the desired reactions in CollDetect_UserAction. The precondition for this is user group “Expert” or higher.

The program CollDetect_UserAction is called by the robot controller by means of the $STOPMESS interrupt. The restrictions that apply to the relevant interrupt programs must therefore be observed during programming.

Precondition

The general preconditions for collision detection are:

- $IMPROVED_COLLMON == TRUE (this is the default setting.)
- $ADAP_ACC ≠ #NONE (this is the default setting.)
- The load data are correct.

TORQMON (old)

In programs from earlier KSS versions, a torque monitoring function may still be used that is programmed using the inline form **TORQMON SetLimits**. This monitoring function is essentially still operational and works in the same way as before.

The precondition for this, however, is as follows: $IMPROVED_COLLMON == FALSE. This means that the current version of the collision detection function is not available.

Either the previous torque monitoring function (TORQMON) or the current collision monitoring function can be used, but not both. It is not possible to use both functionalities at the same time.

Even if $IMPROVED_COLLMON == FALSE, the previous inline form TORQMON can no longer (!) be programmed.

8.14.2 Resuming motion after a collision

Acknowledge message

If the message *Ackn.: Collision detection axis {Axis number}* is active, it must be acknowledged before the robot can be moved again. The signal

$COLL_ALARM is set to FALSE again when $STOPMESS is no longer active.

Program mode

Resuming motion in program mode:

If program mode is resumed after a detected collision (via “Start” or “Start backwards”), detection is immediately active again.

Jog mode

Resuming motion in jog mode:

If jogging is carried out after a detected collision, the detection is automatically suspended for 60 ms.

Safe retraction

Following a collision, the forces and torques acting on the robot axes may be so great that the detection function permanently prevents resumption of motion. The user must safely retract the robot by hand, i.e. move it out of the collision position.

The following options exist for safe retraction of the robot:

- Backward motion using the jog keys (**Jog options**, **Track** option)

  Collision detection is automatically canceled for 1 second. The robot moves back along the same path that it had previously executed.
- **Jog options**, **Override collision detection** option

  The user can override the collision detection function via a check box, i.e. deactivate it. It remains inactive until it is reactivated via the check box.

The **Track** option should be used for preference for safe retraction. Only use **Override collision detection** if **Track** cannot be used, e.g. if the robot is jammed following the collision.

8.14.3 Configuring collision detection for jogging

Description

Collision detection for jogging is inactive if the corresponding values for all axes are “0”. This is the default setting.

In order to use the collision detection function, values must be set. The values can be determined or entered directly.

Precondition

- User rights: Function group **Calibration**

Procedure:
Determination

1. In the main menu, select **Configuration** > **Collision detection** > **Jogging configuration**.

   The **Collision detection - Jogging configuration** window opens.
2. Reset the previous peak values to zero.
3. Jog the robot manually. Use axis-specific and Cartesian jogging, moving all axes in all directions. During jogging, the values in the **Peak value** column change.

Jog the robot until the values in the **Peak value** column no longer change.

> It is important for the **Collision detection - Jogging configuration** window to be open while the robot is moved. Only then are the peak values also determined for the axes for which collision detection is (still) inactive.

4. Apply the peak values in the **Default values** column.
5. Save.

Procedure:
Direct entry

1. In the main menu, select **Configuration** > **Collision detection** > **Jogging configuration**.
   The **Collision detection - Jogging configuration** window opens.
2. Enter the desired values manually in the **Default values** column.
3. Save.

> The sensitivity of the collision detection function can additionally be increased or decreased in the **Jog options** window by means of the **Default value offset** setting.

8.14.4 “Collision detection - Jogging configuration” window

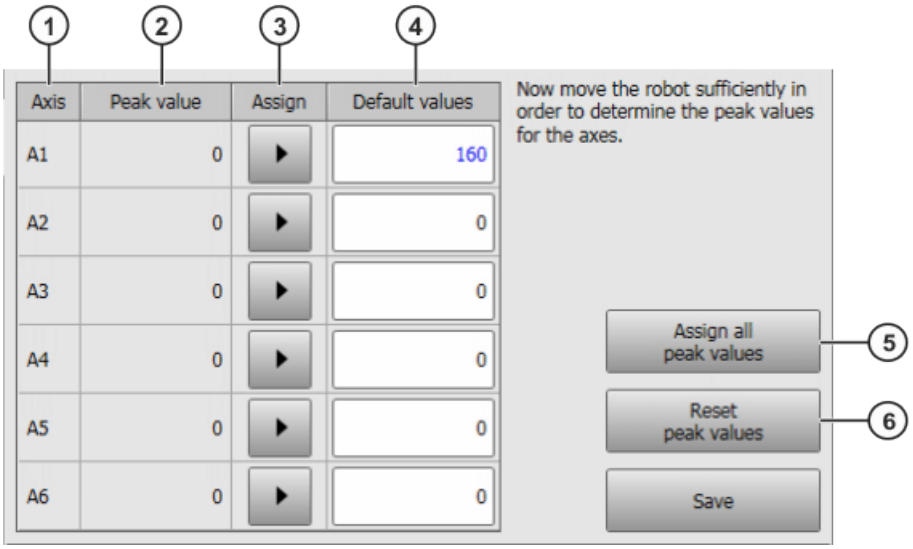


Fig. 8-20: Collision detection - Jogging configuration

Item	Description
1	Axis number
2	Values continuously determined by the robot controller in the background while the axes are moving. The highest value determined since the last reset is displayed here ($COLL-MON_MAX[]). The user can reset the values to zero. • 0 … 500
	Calculation runs in both jog mode and program mode. Precondition for determining values: • Collision detection is active. • Alternatively in jog mode: The **Collision detection - Jogging configuration** window is open.
3	Transfers the value from the **Peak value** box to the **Default values** box and adds an internally defined offset.
4	Default value for collision detection in jog mode. The value can be changed. • 0 … 500 The lower the value, the more sensitively the detection responds. “0” means that detection is deactivated for this axis. Default: 0.
5	Transfers all values from the **Peak value** column to the **Default values** column and adds an internally defined offset.
6	Sets the peak values for all axes to zero.

8.14.5 Configuring collision detection for program mode / filling data sets

To activate collision detection for a motion, the user selects a data set in the inline form (**CDSet_Set[***No.***]**). A maximum of 30 data sets are available.

The data sets contain default values. For meaningful collision detection, the data sets must be configured, i.e. filled with suitable values.

The following methods are available for filling the data sets:

- Via the smartHMI: direct entry of the values
- Via the smartHMI: accepting the values determined by the robot controller
- By means of a program run with the inline form “SaveMax”

8.14.5.1 Filling data sets via smartHMI

Description

The values can be entered directly. Or, if the robot controller has already determined values, these values can be applied.

Precondition

- User rights: Function group **Calibration**
- Only if values are to be applied:

  A suitable motion program must first have been executed with active collision detection in order for the robot controller to be able to determine peak values.

Procedure

1. In the main menu, select **Configuration** > **Collision detection** > **Data set configuration**.

   The **Collision detection - Data set configuration** window opens.
2. Select a data set.
3. If required, assign a name for the data set.
4. Either: Enter the desired values directly in the **Data set values** column.

   Or: Apply the peak values in the **Data set values** column.
5. Save.

8.14.5.2 “Collision detection - Data set configuration” window

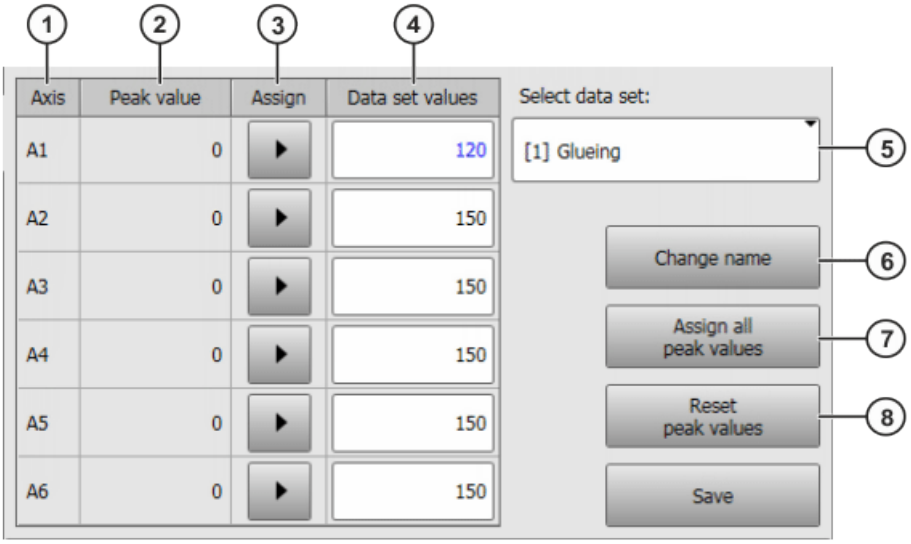


Fig. 8-21: Collision detection - Data set configuration

Item	Description
1	Axis number
2	Values continuously determined by the robot controller in the background while the axes are moving. The highest value determined since the last reset is displayed here ($COLL-MON_MAX[]). The user can reset the values to zero. • 0 … 500
	Calculation runs in both jog mode and program mode. Precondition for determining values: • Collision detection is active. • Alternatively in jog mode: The **Collision detection - Jogging configuration** window is open.
3	Transfers the value from the **Peak value** box to the **Data set values** box and adds an internally defined offset.

Item	Description
4	Default value for collision detection in program mode. The value can be changed. • 0 … 500 The lower the value, the more sensitively the detection responds. “0” means that detection is deactivated for this axis. Default: 0.
5	Select the data set whose values are to be displayed in the table.
6	The name of the data set can be changed (max. 24 characters).
7	Transfers all values from the **Peak value** column to the **Data set values** column and adds an internally defined offset.
8	Sets the peak values for all axes to zero.

8.14.5.3 Filling data sets via the program with “SaveMax”

Description

The data sets for collision detection can be filled with suitable values via the inline form “SaveMax”. This is done by means of a specially executed program run. The values for collision detection are then available.

The inline form “SaveMax” reads $COLLMON_MAX[]. In order to obtain the specific values that occur in a specific motion sequence, $COLLMON_MAX[] must therefore be set to “0” at the start of the motion sequence.

Precondition

- User group “Expert” or higher
- Program is selected or open.

Procedure

The following procedure illustrates one method for determining a suitable value for the collision detection for a program section.

1. In the program, set $COLLMON_MAX[1] to [6] to *0*.

   This must be carried out before the first motion for which a value is to be determined.
2. Following the last motion for which a value is to be determined, insert the inline form SaveMax:

   Select the menu sequence **Commands**> **Motion parameters** > **Collision detection**. In the inline form, select the entry **SaveMax**.
3. Set the parameters in the inline form SaveMax:
   - Enter the desired value under **Offset**. Recommendation: between “10” and “20”.
   - Under **DataSet**, select the data set **CDSet_Set[***No.***]** in which the value is to be saved.
4. For all motions for which a value is to be determined, set the parameter **ColDetect** to **CDSet_Set[***No.***]**. Select the same number as under **DataSet** in SaveMax.
5. Execute the program with the programmed velocity.
6. Select or open the program again.

- Delete the lines in which $COLLMON_MAX[1] to [6] is set to the value “0”.
- Delete the inline form SaveMax.

> It is necessary to activate collision detection in the inline forms, as described, already for the determination run. During the first determination run, the default tolerance value of 150% is valid for all axes.
> This default value is generally sufficient to execute the determination run without triggering collision detection. If detection is triggered, however:
> 1. In the **Collision detection - Data set configuration** window, set all values in the **Data set values** column to “500”.
> 2. Perform the determination run again.

8.14.5.4 Inline form COLLDETECT SaveMax

Description

The inline form SaveMax serves to fill the data set **CDSet_Set[*no.*]** with suitable values for collision detection. This is done by means of a specially executed program run.

Fig. 8-22: Inline form COLLDETECT SaveMax

Item	Description
DataSet	• **CDSet_Set[1]** … **[30]** Select the data set to be filled. For all robot axes, the data set is filled with: $COLLMON_MAX[*axis no.*] + Offset As standard, the data sets contain the value *150* for all axes. When filling using SaveMax, this value is overwritten.
Offset	Recommendation: between *10* and *20* Unit: %

8.14.6 Activating/deactivating collision detection via inline form

Description

To activate collision detection for a motion, the user selects a data set in the inline form. The data set contains the value for collision detection.

Spline:

Within a spline block, the spline segments inherit the setting of the block as standard. The user has the option, however, of making individual settings for each segment: For this, display the **ColDetect** box via **Switch parameter** > **Collision detection** and select the desired data set.

- If the **ColDetect** box is displayed in the inline form of the segment, the setting in this box is valid for the segment.
- If the **ColDetect** box is hidden in the inline form of the segment, the setting at the block is valid for the segment.

Precondition

- A program is selected.

- T1 mode
- Collision detection has been configured for program mode, i.e. at least one data set has been filled with suitable values and it is known which data set this is (number or name).

Procedure for activation

1. If the box **ColDetect** is not shown in the inline form, it can be displayed via **Switch parameter** > **Collision detection**.
2. Select the data set under **ColDetect** in the inline form.

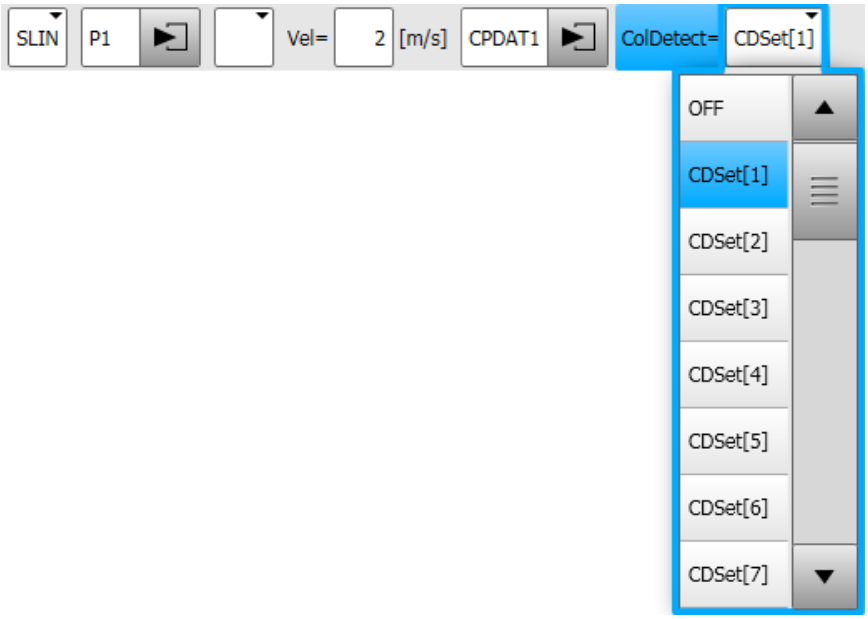


Fig. 8-23: Inline form with ColDetect, SLIN example

3. Confirm the selection with **Cmd OK**.

In the editor, the statement now contains the addition `ColDetect [No.]`.

```
SLIN P1 Vel=2 m/s CPDAT1 Tool[1] Base[1] ColDetect[1]
```

Fig. 8-24: Inline statement with ColDetect, SLIN example

Procedure for deactivation

To switch collision detection off again for a motion, select the OFF setting under **ColDetect** in the inline form.

8.14.7 Displaying current values / Collision detection - Display window

Procedure

- In the main menu, select **Configuration** > **Collision detection** > **View**.

  The **Collision detection - Display** window opens.

Description

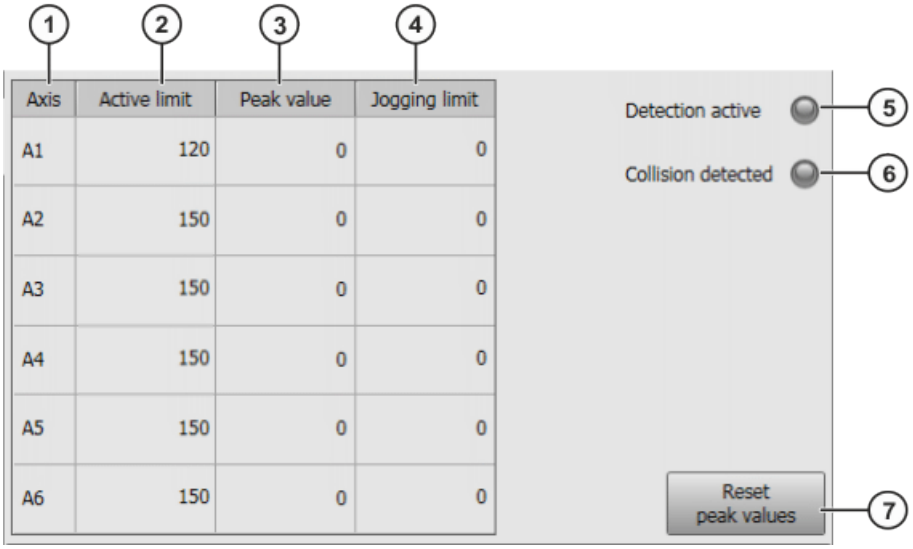


Fig. 8-25: Collision detection - Display

Item	Description
1	Axis number
2	The currently active limit for each axis. This can either be the limit for jog mode or the limit for program mode. The lower the value, the more sensitively the detection responds. “0” means that detection is deactivated for this axis.
3	The identical values that are also displayed in the same columns in the **Collision detection - Jogging configuration** and **Collision detection - Data set configuration** windows. The user can reset the values to zero. • 0 … 500
4	The current values for collision detection in jog mode • The values are based on the **Default values** column from the **Collision detection - Jogging configuration** window. • Additionally, the factor **Default value offset** from the **Jog options** window is taken into consideration.
5	• Gray: Collision detection is not active. • Green: Collision detection is active.
6	• Gray: No collision. • Red: The controller has detected a collision.
7	Sets the peak values for all axes to zero.

8.14.8 Collision detection – System variables

Precondition

The general preconditions for collision detection are:

- $IMPROVED_COLLMON == TRUE (this is the default setting.)
- $ADAP_ACC ≠ #NONE (this is the default setting.)
- The load data are correct.

Collision detection can be configured via the smartHMI (main menu **Configuration** > **Collision detection** and inline form COLLDETECT). Alternatively, it can be configured via the system variables. **Important:** Do not combine the two options.
Configuration of the collision detection function via the smartHMI is recommended.

System variables

System variable	Description
$IMPROVED_COLLMON	• **TRUE** (default): – The collision detection function is available. This does not mean that it is automatically active. Whether or not it is active depends on how the other system variables are set. – The previous torque monitoring function option (TORQMON) is not available. • **FALSE**: – Programs containing the previous torque monitoring function (TORQMON) behave as previously. The corresponding system variables are available and behave as previously. The previous inline form TORQMON can no longer (!) be programmed. – The current collision detection function is not available. The main menu items under **Configuration** > **Collision detection** cannot be called. The system variables are not available or have a different meaning. The variable can be modified in KRC:\STEU\Mada\$custom.dat. The variable cannot be modified via the variable correction function.
$COLLMON_ACTIVE_PRO	Collision detection for program mode • **TRUE** (default): Active • **FALSE**: Not active
$COLLMON_ACTIVE_COM	Collision detection for jog mode • **TRUE** (default): Active • **FALSE**: Not active

System variable	Description
$COLL-MON_TOL_PRO_DEF[1] … [12]	Default value for sensitivity of collision detection in program mode. In the case of a cold start, program selection or program reset. $COLLMON_TOL_PRO is automatically set to the value of $COLLMON_TOL_PRO_DEF. • 0 … 500 “0” means that collision detection is deactivated for this axis. Default: 0 The variable can be modified in KRC:\STEU\Mada\$custom.dat. The variable cannot be modified via the variable correction function.
	Value visible and editable in: **Collision detection - Data set configuration** window, **Data set values** column
$COLL-MON_TOL_COM_DEF[1] … [12]	Default value for sensitivity of collision detection in jog mode. In the case of a cold start, $COLLMON_TOL_COM is automatically set to the value of $COLLMON_TOL_COM_DEF. • 0 … 500 “0” means that collision detection is deactivated for this axis. Default: 0 The variable can be modified in KRC:\STEU\Mada\$custom.dat. The variable cannot be modified via the variable correction function.
	Value visible and editable in: **Collision detection - Jogging configuration** window, **Default values** column
$COLLMON_TOL_PRO[1] … [12]	Value for collision detection in program mode • 0 … 500 “0” means that collision detection is deactivated for this axis. Default: 0
$COLLMON_TOL_COM[1] … [12]	Value for collision detection in jog mode • 0 … 500 $COLLMON_TOL_COM is based on $COLL-MON_TOL_COM_DEF. Additionally, $COLLMON_TOL_COM takes into consideration the factor **Default value offset** from the **Jog options** window. “0” means that collision detection is deactivated for this axis. Default: 0
	Value visible in: **Collision detection - Display** window, **Jogging limit** column

System variable	Description
$COLLMON_TOL_ACT[1] … [12]	Currently valid value for collision detection • In program mode: $COLLMON_TOL_PRO • In jog mode: $COLLMON_TOL_COM The variable is write-protected.
	Value visible in: **Collision detection - Display** window, **Active limit** column
$COLLMON_MAX[1] … [12]	$COLLMON_MAX contains the highest value that has occurred for each axis since the last reset. The robot controller continuously determines the values in the background while the axes are moving. Precondition for determining values: $COLLMON_TOL_ACT[1] ≠ 0! The user can set $COLLMON_MAX to “0”. Reading from or writing to this variable triggers an advance run stop. This can be prevented with CONTINUE.
	Value visible in the following windows, in the **Peak value** column in each case: • **Collision detection - Jogging configuration** • **Collision detection - Data set configuration** • **Collision detection - Display**
$COLL_ENABLE	The status of this signal is indicated in the **Collision detection - Display** window by the **Detection active** LED. • **TRUE**: Collision detection is active. (LED: green) • **FALSE**: Collision detection is not active. (LED: gray) The variable is write-protected.
$COLL_ALARM	Signal that is set to TRUE if a collision has been detected. The signal remains set as long as $STOPMESS is active. The status is indicated in the **Collision detection - Display** window by the **Collision detected** LED. • **TRUE**: The controller has detected a collision. (LED: red) • **FALSE**: No collision. (LED: gray) The variable is write-protected.

9 Basic principles of motion programming

9.1 Overview of motion types

The following motion types can be programmed:

- **Point-to-point motion (PTP)**
  (>>> *9.2 "Motion type PTP" Page 355*)
- **Linear motions (LIN)**
  (>>> *9.3 "Motion type LIN" Page 356*)
- **Circular motion (CIRC)**
  (>>> *9.4 "Motion type CIRC" Page 356*)
- **Spline motions**
  Spline motions have a number of advantages over conventional PTP, LIN and CIRC motions.
  (>>> *9.7 "Spline motion type" Page 361*)

> The start point of a motion is always the end point of the previous motion.

The following motions are known as CP (“Continuous Path”) motions.

- LIN, CIRC, CP spline blocks, SLIN, SCIRC

9.2 Motion type PTP

The robot guides the TCP along the fastest path to the end point. The fastest path is generally not the shortest path and is thus not a straight line. As the motions of the robot axes are rotational, curved paths can be executed faster than straight paths.

The exact path of the motion cannot be predicted.

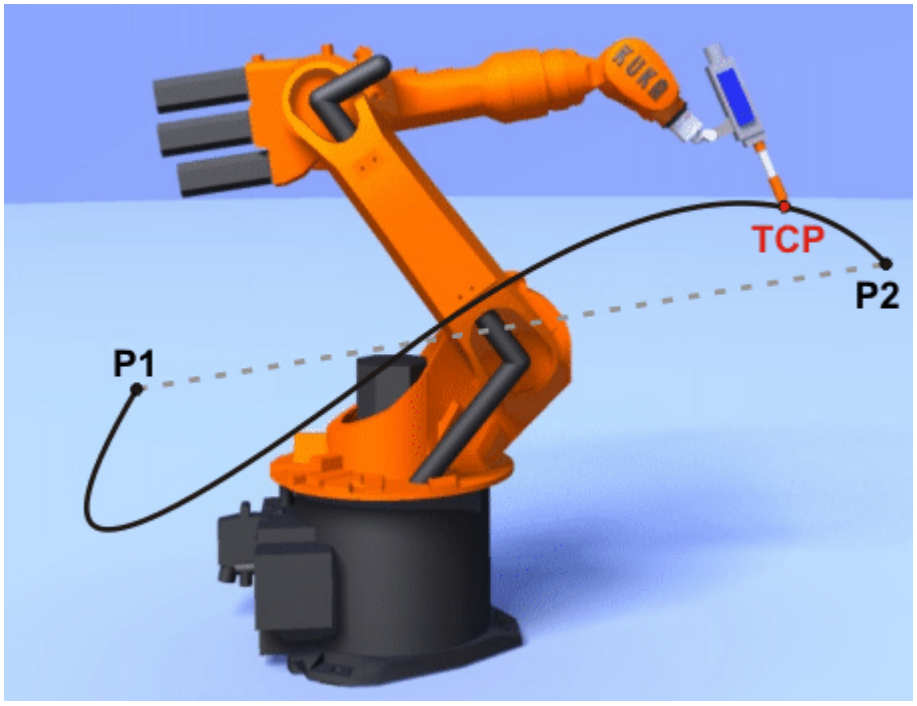


Fig. 9-1: PTP motion

9.3 Motion type LIN

The robot guides the TCP at a defined velocity along a straight path to the end point.

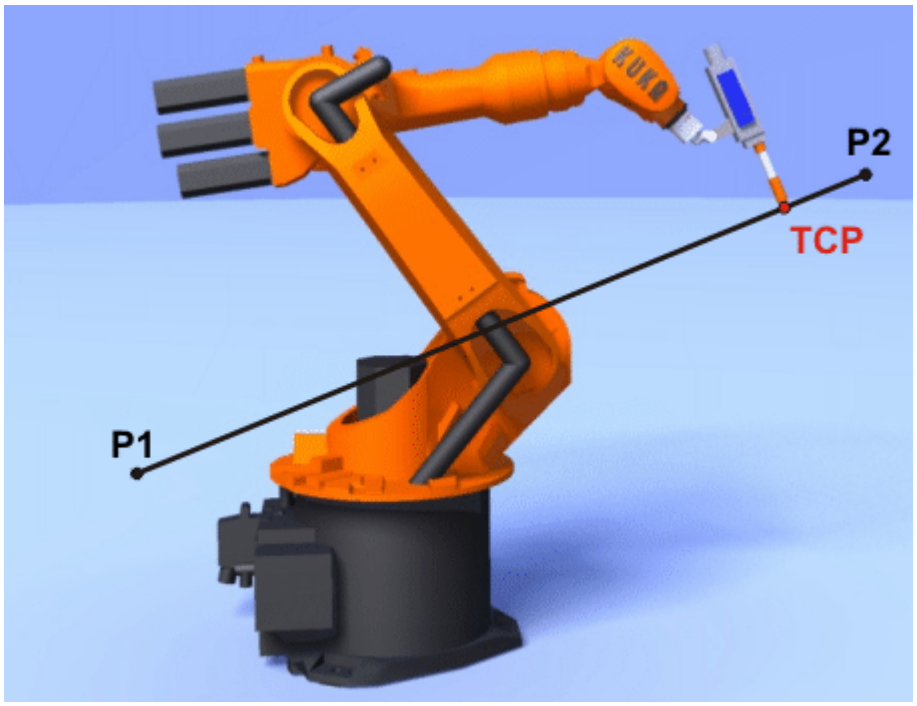


Fig. 9-2: LIN motion

9.4 Motion type CIRC

The robot guides the TCP at a defined velocity along a circular path to the end point. The circular path is defined by a start point, auxiliary point and end point.

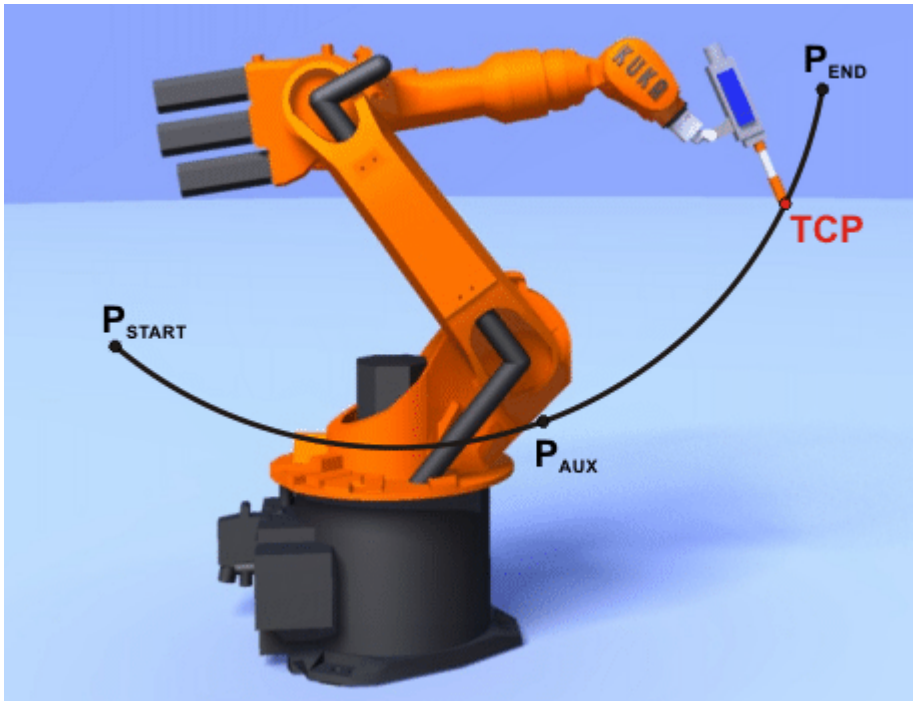


Fig. 9-3: CIRC motion

9.5 Approximate positioning

Approximate positioning means that the motion does not stop exactly at the programmed point. Approximate positioning is an option that can be selected during motion programming.

Approximate positioning is not possible if the motion instruction is followed by an instruction that triggers an advance run stop.

Approximate positioning with a PTP motion

The TCP leaves the path that would lead directly to the end point and moves along a faster path. During programming of the motion, the maximum distance from the end point at which the TCP may deviate from its original path is defined.

The path of an approximated PTP motion cannot be predicted. It is also not possible to predict on which side of the approximated point the path will run.

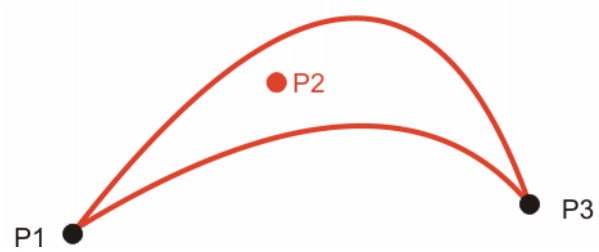


Fig. 9-4: PTP motion, P2 is approximated

Approximate positioning with a LIN motion

The TCP leaves the path that would lead directly to the end point and moves along a shorter path. During programming of the motion, the maximum distance from the end point at which the TCP may deviate from its original path is defined.

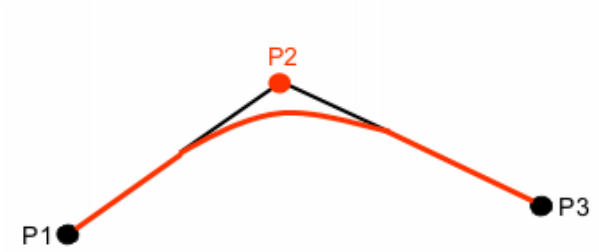


Fig. 9-5: LIN motion, P2 is approximated

Approximate positioning with a CIRC motion

The TCP leaves the path that would lead directly to the end point and moves along a shorter path. During programming of the motion, the maximum distance from the end point at which the TCP may deviate from its original path is defined.

The motion passes exactly through the auxiliary point.

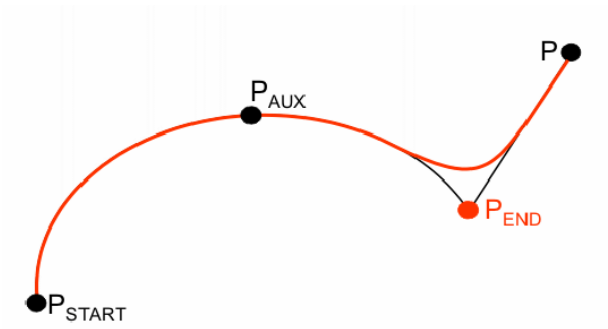


Fig. 9-6: CIRC motion, P_{END} is approximated

9.6 Orientation control LIN, CIRC

Description

The orientation of the TCP can be different at the start point and end point of a motion. There are several different types of transition from the start orientation to the end orientation. A type must be selected when a CP motion is programmed.

The orientation control for LIN and CIRC motions is defined as follows:

- In the option window **Motion parameters**
  (>>> *10.3.8 "Option window “Motion parameters” (LIN, CIRC, PTP)" Page 396*)
- Or via the system variable $ORI_TYPE

LIN motion

Orientation control	Description
• Option window: **Constant orientation** • $ORI_TYPE = #CONSTANT	The orientation of the TCP remains constant during the motion. The programmed orientation is disregarded for the end point and that of the start point is retained.
• Option window: **Standard** • $ORI_TYPE = #VAR	The orientation of the TCP changes continuously during the motion. **Note**: If, with **Standard**, the robot passes through a wrist axis singularity, use **Wrist PTP** instead.
• Option window: **Wrist PTP** • $ORI_TYPE = #JOINT	The orientation of the TCP changes continuously during the motion. This is done by linear transformation (axis-specific motion) of the wrist axis angles. **Note**: Use **Wrist PTP** if, with **Standard**, the robot passes through a wrist axis singularity. The orientation of the TCP changes continuously during the motion, but not uniformly. **Wrist PTP** is thus not suitable if a specific orientation must be maintained exactly, e.g. in the case of laser welding.

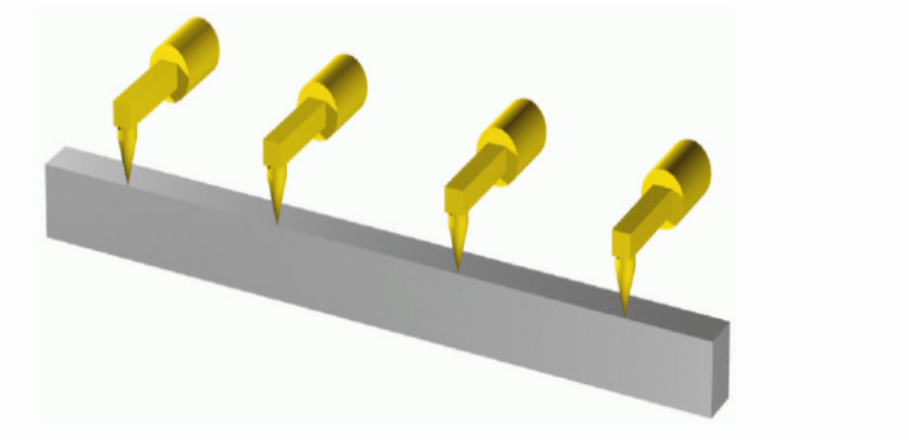


Fig. 9-7: Constant orientation

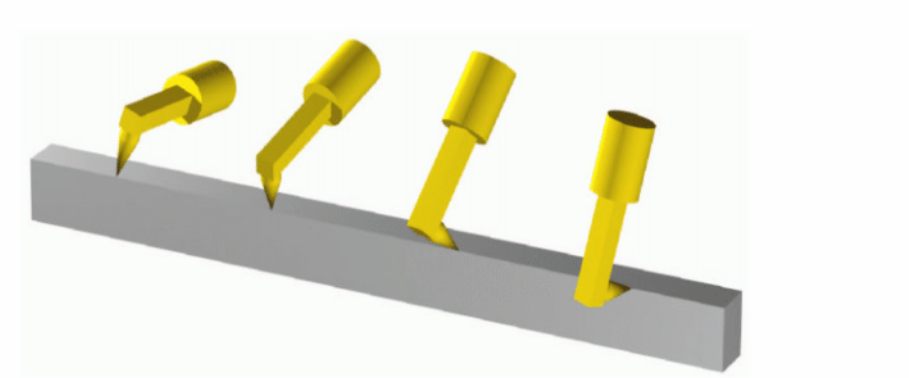


Fig. 9-8: Standard or Wrist PTP

CIRC motion

During CIRC motions, the robot controller only takes the programmed orientation of the end point into consideration. The programmed orientation of the auxiliary point is disregarded.

The same orientation control options are available for selection for CIRC motions as for LIN motions.

It is also possible to define for CIRC motions whether the orientation control is to be base-related or path-related. This is defined via the system variable $CIRC_TYPE.

Orientation control	Description
$CIRC_TYPE = #BASE	Base-related orientation control during the circular motion
$CIRC_TYPE = #PATH	Path-related orientation control during the circular motion

$CIRC_TYPE is meaningless if $ORI_TYPE = #JOINT.

(>>> *9.6.1 "Combinations of $ORI_TYPE and $CIRC_TYPE" Page 359*)

9.6.1 Combinations of $ORI_TYPE and $CIRC_TYPE

$ORI_TYPE = #CONSTANT, $CIRC_TYPE = #PATH:

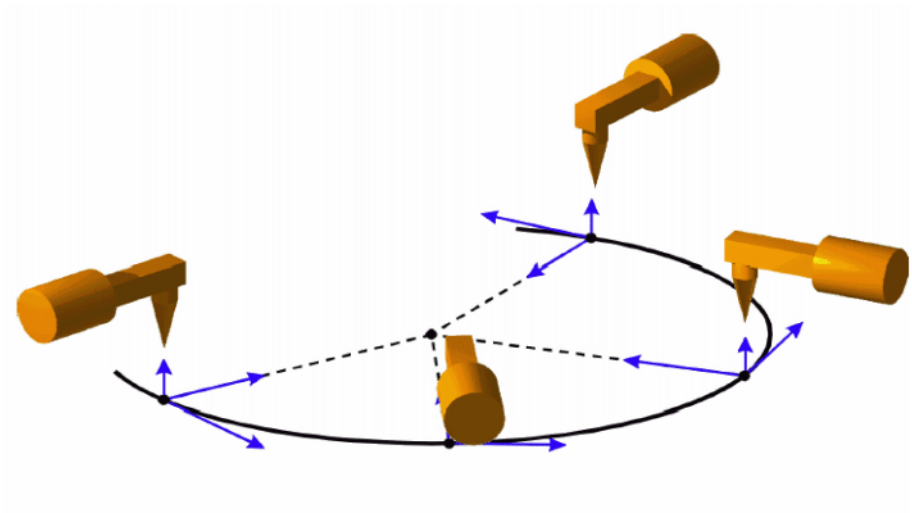


Fig. 9-9: Constant orientation, path-related

$ORI_TYPE = #VAR, $CIRC_TYPE = #PATH:

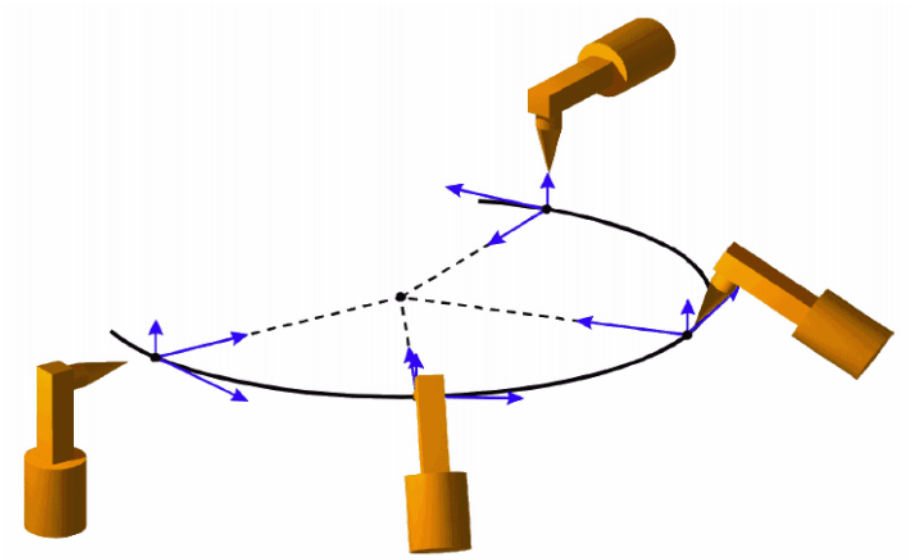


Fig. 9-10: Variable orientation, path-related

$ORI_TYPE = #CONSTANT, $CIRC_TYPE = #BASE:

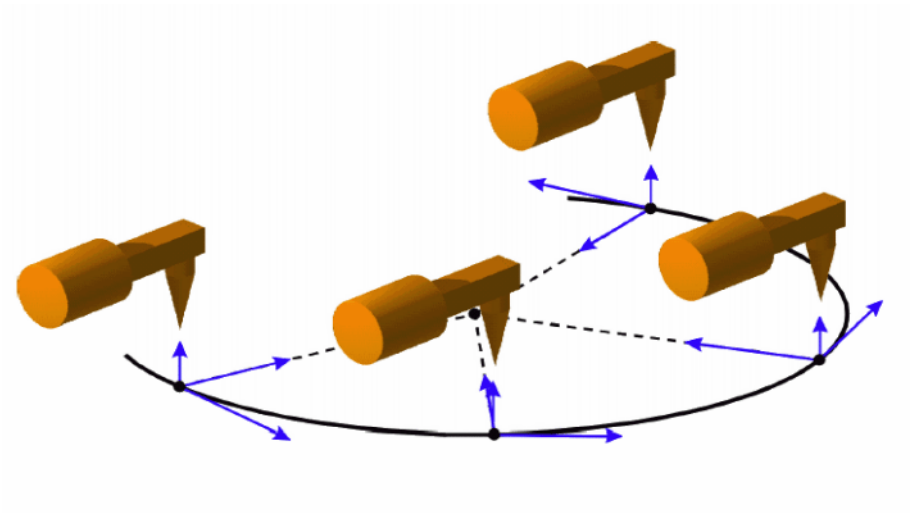


Fig. 9-11: Constant orientation, base-related

$ORI_TYPE = #VAR, $CIRC_TYPE = #BASE:

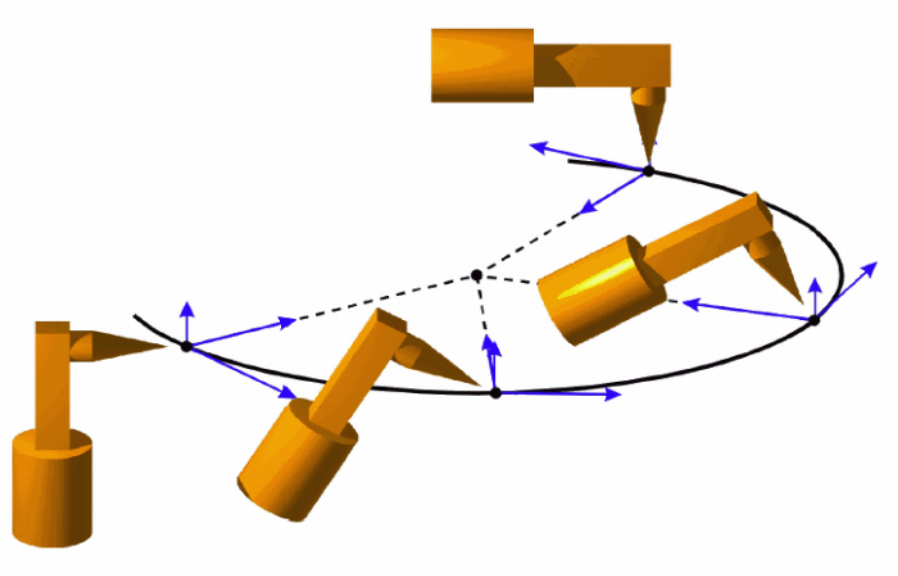


Fig. 9-12: Variable orientation, base-related

9.7 Spline motion type

Spline is a motion type that is particularly suitable for complex, curved paths. Such paths can also be generated using approximated LIN and CIRC motions, but splines have advantages, however.

The most versatile spline motion is the spline block. A spline block is used to group together several motions as an overall motion. The spline block is planned and executed by the robot controller as a single motion block.

The motions that may be included in a spline block are called spline segments. They are taught separately.

- A CP spline block can contain SPL, SLIN and SCIRC segments.
- A PTP spline block can contain SPTP segments.

In addition to spline blocks, individual spline motions can also be programmed: SLIN, SCIRC and SPTP.

Advantages of spline blocks

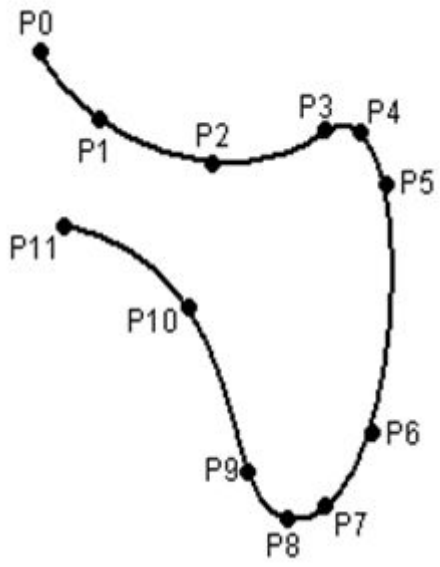


Fig. 9-13: Curved path with spline block

- The path is defined by means of points that are located on the path. The desired path can be generated easily.
- The programmed velocity is maintained better than with conventional motion types. There are few cases in which the velocity is reduced.

  (>>> *9.7.1 "Velocity profile for spline motions" Page 363*)

  Furthermore, special constant velocity ranges can be defined in CP spline blocks.
- The path always remains the same, irrespective of the override setting, velocity or acceleration.
- Circles and tight radii are executed with great precision.

Disadvantages of LIN/CIRC

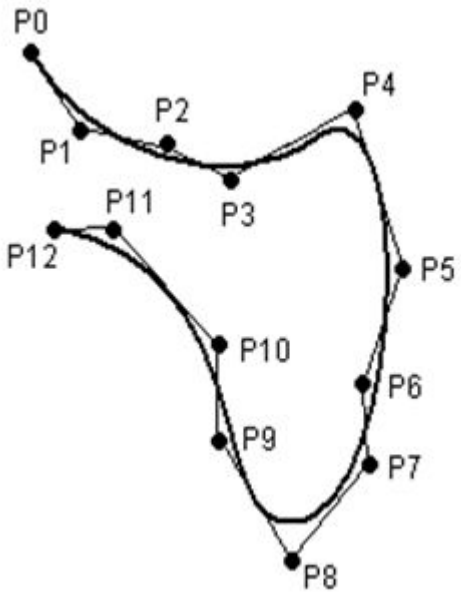


Fig. 9-14: Curved path with approximated LIN motions

- The path is defined by means of approximated points that are not located on the path. The approximate positioning ranges are difficult to

predict. Generating the desired path is complicated and time-consuming.

- In many cases, the velocity may be reduced in a manner that is difficult to predict, e.g. in the approximate positioning ranges and near points that are situated close together.
- The path changes if approximate positioning is not possible, e.g. for time reasons.
- The path changes in accordance with the override setting, velocity or acceleration.

9.7.1 Velocity profile for spline motions

The path always remains the same, irrespective of the override setting, velocity or acceleration.

The robot controller already takes the physical limits of the robot into consideration during planning. The robot moves as fast as possible within the constraints of the programmed velocity, i.e. as fast as its physical limits will allow. This is an advantage over conventional LIN and CIRC motions for which the physical limits are not taken into consideration during planning. It is only during motion execution that these limits have any effect and can cause stops to be triggered.

Reduction of the velocity

Prime examples of cases in which the velocity has to fall below the programmed velocity include:

- Tight corners
- Major reorientation
- Large motions of the external axes
- Motion in the vicinity of singularities

Reduction of the velocity due to major reorientation can be avoided with spline segments by selecting the orientation control option **Ignore orientation**.

Reduction of the velocity due to major external axis motions can be avoided for spline segments with $EX_AX_IGNORE.

Reduction of the velocity to 0

This is the case for:

- Successive points with the same coordinates.
- Successive SLIN and/or SCIRC segments. Cause: inconstant velocity direction.

  In the case of SLIN-SCIRC transitions, the velocity is also reduced to 0 if the straight line is a tangent of the circle, as the circle, unlike the straight line, is curved.

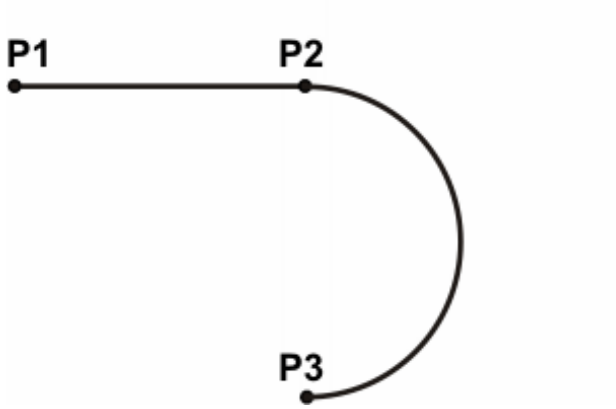


Fig. 9-15: Exact positioning at P2

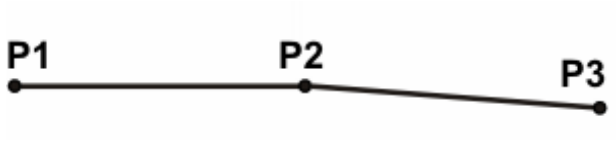


Fig. 9-16: Exact positioning at P2

Exceptions:

- In the case of successive SLIN segments that result in a straight line and in which the orientations change uniformly, the velocity is not reduced.

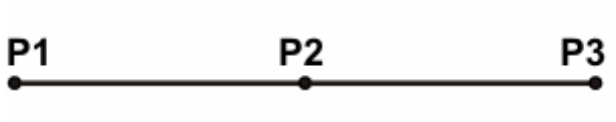


Fig. 9-17: P2 is executed without exact positioning.

- In the case of a SCIRC-SCIRC transition, the velocity is not reduced if both circles have the same center point and the same radius and if the orientations change uniformly. (This is difficult to teach, so calculate and program points.)

Circles with the same center point and the same radius are sometimes programmed to obtain circles ≥ 360°. A simpler method is to program a circular angle.

9.7.2 BCO run with spline motions via the Block selection button

Spline block

Block selection can be made to the segments of a spline block.

- CP spline block: The BCO run is executed as a conventional LIN motion.
- PTP spline block: The BCO run is executed as a conventional PTP motion.

Following a block selection, the path is generally the same as if the spline were to be executed during normal program execution.

Exceptions are possible if the spline has never yet been executed prior to the block selection and the block selection is made to the start of the spline block.

The start point of the spline motion is the last point before the spline block, i.e. the start point is outside the block. The robot controller saves the start point during normal execution of a spline. In this way, it is already known in the case of a block selection being carried out subsequently. If the spline block has never yet been executed, however, the start point is unknown.

If the Start key is pressed after the BCO run, the modified path is indicated by means of a message that must be acknowledged.

Example: modified path in the case of block selection to P1 with unknown P0

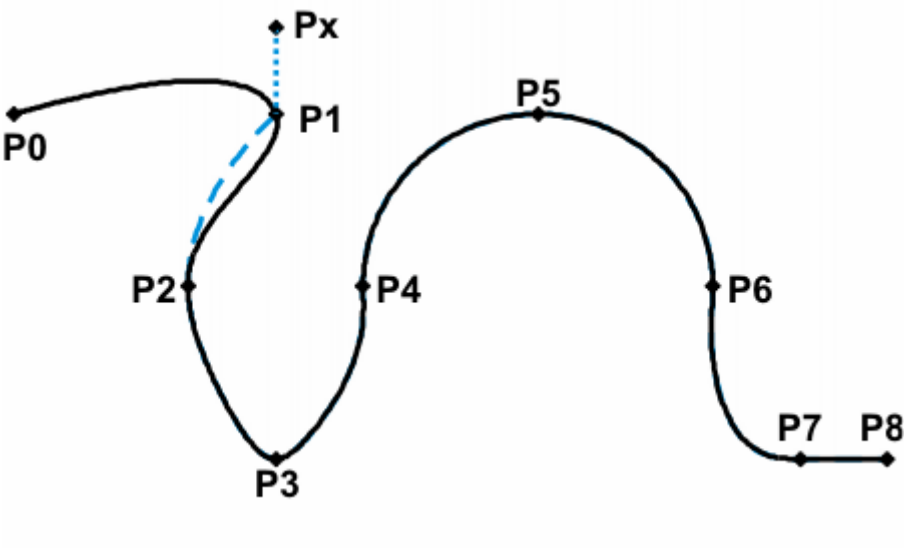


Fig. 9-18: Example: modified path in the case of block selection to P1 with unknown P0

```
 1 PTP P0
 2 SPLINE
 3     SPL P1
 4     SPL P2
 5     SPL P3
 6     SPL P4
 7     SCIRC P5, P6
 8     SPL P7
 9     SLIN P8
10 ENDSPLINE
```

Line	Description
1	Last point before the spline block = start point of the spline motion
2	Header/start of the CP spline block
3 … 9	Spline segments
10	End of the CP spline block

SCIRC

In the case of block selection to a SCIRC (segment or individual block) for which a circular angle has been programmed, the motion is executed to the end point, taking into consideration the circular angle, provided that the robot controller knows the start point.

If the robot controller does not know the start point, the motion is executed to the programmed end point. In this case, a message is generated, indicating that the circular angle is not being taken into consideration.

9.7.3 BCO run with spline motions after program modification

The following cases can be distinguished here:

- Modification to the current spline block, i.e. to the spline block in which the main run is currently located
- Modification to a spline block other than the current one

The following applies to both cases:

- Modification to CP spline block: The BCO run is executed as a conventional LIN motion.
- Modification to PTP spline block: The BCO run is executed as a conventional PTP motion.

9.7.3.1 BCO run following modification to the current spline block

BCO run to end point

BCO run to an end point

The following modifications to the current spline block lead to a BCO run to an end point:

- Adding a motion: BCO run to the end point of the new motion
- Deleting the current segment:
  - BCO run to the end point of the previous segment
  - Or, if the deleted motion was the first in the block: BCO run to the spline header
- Reteaching a point: BCO run to the modified point

  In this case, the robot is already at the target in Cartesian terms. It will only move during the BCO run if further parameters have been modified (e.g. orientation).

BCO run to the path

BCO run to the nearest position on the path

Following certain modifications, the robot controller executes a BCO run to the position on the path which is closest to the current position. In other words, a programmed point is not addressed, as is usually the case for BCO runs, but rather the nearest position on the path is. This occurs with the following modifications:

- Deleting a segment other than the current one from the current block
- Adding, modifying or deleting the following elements:
  - Logic statements (e.g. trigger or time block)
  - Parameters influencing motion (e.g. velocity or orientation)
  - Circular angle

Behavior after modification of the circular angle:

- If the circular angle has been modified in such a way that the robot is then still on the circular path in Cartesian terms, it stays at this position during the BCO run. It will only move during the BCO run if further parameters have been modified (e.g. orientation).

- If the circular angle has been modified in such a way that the robot is then no longer on the circular path in Cartesian terms, a LIN-BCO run is executed to the nearest Cartesian position on the circular path.

Example

The example shows a common application – namely, the modification of the velocity of a segment.

```
1 PTP P0
2 SPLINE
3     SPL P1
4     SLIN P2
5     SPL P3
6 ENDSPLINE
```

1. The user stops the program between P2 and P3.
2. He modifies the velocity of segment `SPL P3` and continues the program.
3. The robot controller generates the message *BCO run active* and executes a BCO run to the nearest position on the path. Since the robot is still positioned on the path, it moves only minimally or not at all.
4. The robot controller generates the following message: *BCO reached at {Distance covered in [%]} of the programmed path of {Spline motion type}*

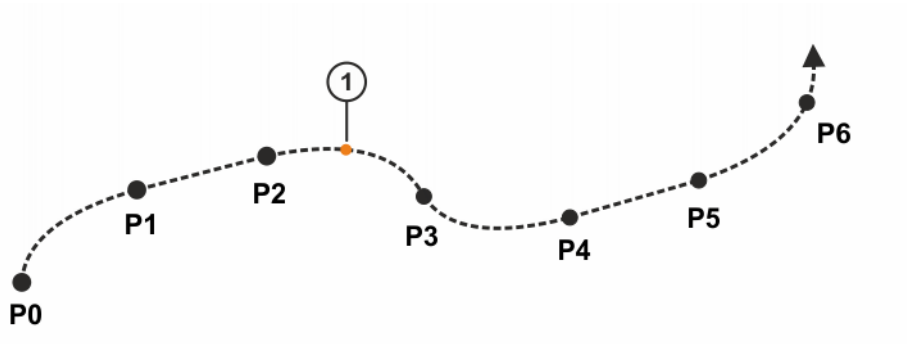


Fig. 9-19: Example

Item	Description
1	Point at which the robot has been stopped The robot was not moved when the velocity is modified. The nearest position on the path is therefore the actual position. As a result, the BCO run goes to the actual position.

9.7.3.2 BCO run following modification of another spline block

Modification to a spline block other than the current one

- Behavior if a segment in another spline block has been modified:

  Upon continuation of the program, the BCO run goes to the end point of the modified segment.
- Behavior if the header of another spline block has been modified:

  Upon continuation of the program, the BCO run goes to the start point of the modified block.

```
 1 PTP P0
 2 SPLINE
 3     SPL P1
 4     SLIN P2
 5     SPL P3
 6 ENDSPLINE
 7 SPLINE
 8     SPL P4
 9     SLIN P5
10     SPL P6
11 ENDSPLINE
```

Example 1

1. The user stops the program between P2 and P3.
2. He modifies the velocity of segment `SLIN P5` and continues the program.
3. The robot controller generates the following message: *BCO run executed as LIN motion*. The user must acknowledge the message.
4. The robot controller generates the message *BCO run active* and executes a BCO run to P5.
5. The robot controller generates the following message: *Programmed path reached (BCO)*

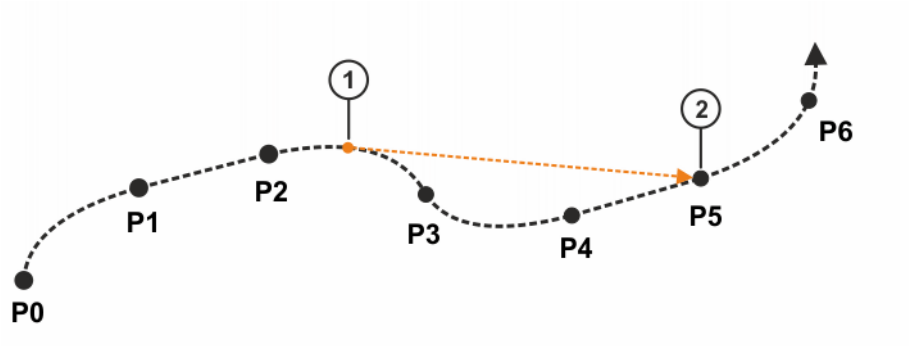


Fig. 9-20: Example 1

Item	Description
1	Point at which the robot has been stopped
2	End point of the modified segment = target of the BCO run

Example 2

1. The user stops the program between P1 and P2.
2. He modifies the velocity of the spline block beginning in line 7 and continues the program.
3. The robot controller generates the following message: *BCO run executed as LIN motion*. The user must acknowledge the message.
4. The robot controller generates the message *BCO run active* and executes a BCO run to the start point of the spline block (in other words, to P3).

   P3 is addressed under the condition that it is already known to the robot controller, i.e. that it has already been executed previously. If P3 is not known, the BCO run goes to P4.
5. The robot controller generates the following message: *Programmed path reached (BCO)*

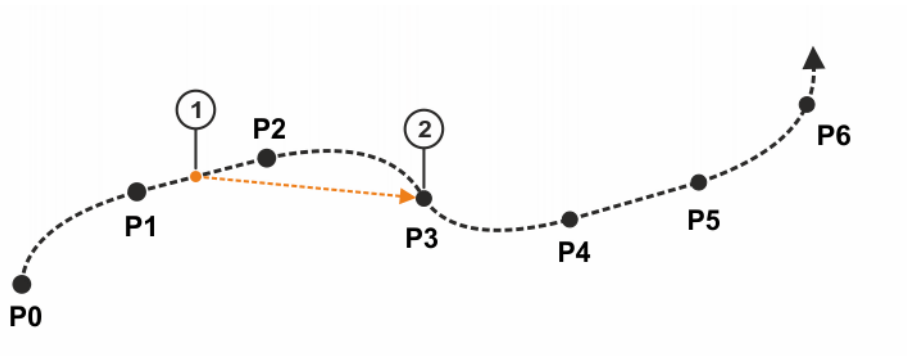


Fig. 9-21: Example 2

Item	Description
1	Point at which the robot has been stopped
2	Start point of the modified spline block = target of the BCO run

9.7.4 Modifications to spline blocks

Description

- Modification of the position of the point:

  If a point within a spline block is offset, the path is modified, at most, in the 2 segments before this point and the 2 segments after it.

  Small point offsets generally result in small modifications to the path. If, however, very long segments are followed by very short segments or vice versa, small modifications can have a very great effect.
- Modification of the segment type:

  If an SPL segment is changed into an SLIN segment or vice versa, the path changes in the previous segment and the next segment.

Example 1

Original path:

```
PTP P0
SPLINE
SPL P1
SPL P2
SPL P3
SPL P4
SCIRC P5, P6
SPL P7
SLIN P8
ENDSPLINE
```

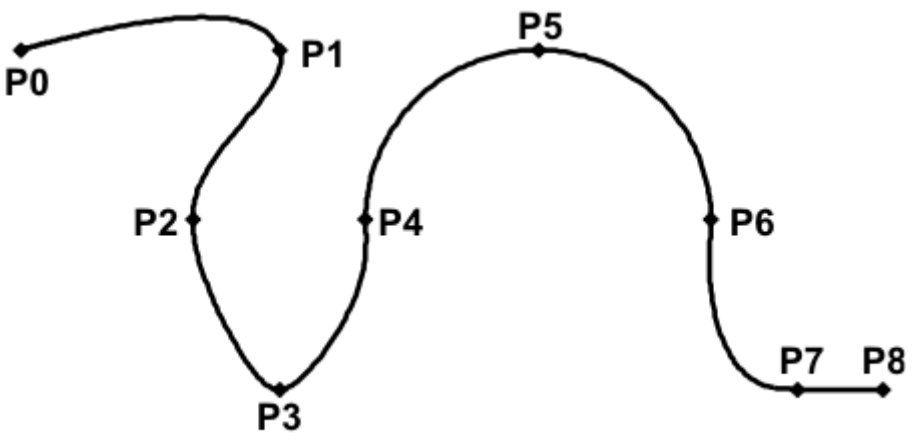


Fig. 9-22: Original path

A point is offset relative to the original path:

P3 is offset. This causes the path to change in segments P1 - P2, P2 - P3 and P3 - P4. Segment P4 - P5 is not changed in this case, as it belongs to an SCIRC and a circular path is thus defined.

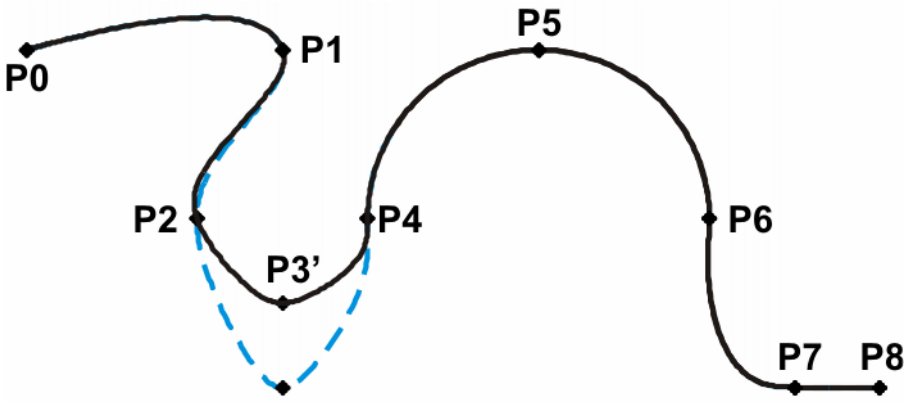


Fig. 9-23: Point has been offset

The type of a segment is changed relative to the original path:

In the original path, the segment type of P2 - P3 is changed from SPL to SLIN. The path changes in segments P1 - P2, P2 - P3 and P3 - P4.

```
PTP P0
SPLINE
SPL P1
SPL P2
SLIN P3
SPL P4
SCIRC P5, P6
SPL P7
SLIN P8
ENDSPLINE
```

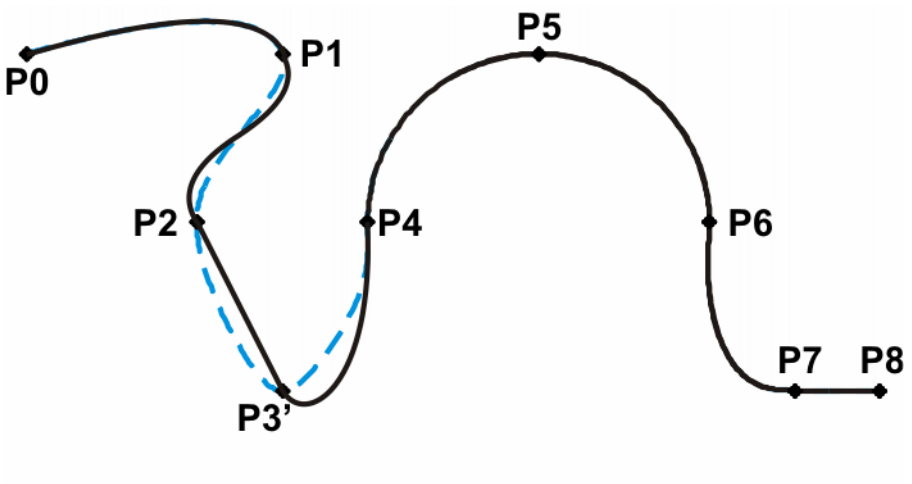


Fig. 9-24: Segment type has been changed

Example 2

Original path:

```
...
SPLINE
SPL {X 100, Y 0, ...}
SPL {X 102, Y 0}
SPL {X 104, Y 0}
SPL {X 204, Y 0}
ENDSPLINE
```

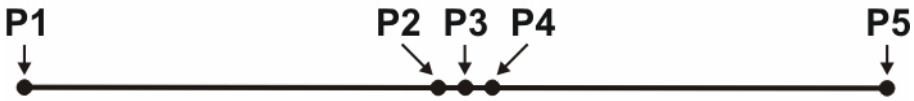


Fig. 9-25: Original path

A point is offset relative to the original path:

P3 is offset. This causes the path to change in all the segments illustrated. Since P2 - P3 and P3 - P4 are very short segments and P1 - P2 and P4 - P5 are long segments, the slight offset causes the path to change greatly.

```
...
SPLINE
SPL {X 100, Y 0, ...}
SPL {X 102, Y 1}
SPL {X 104, Y 0}
SPL {X 204, Y 0}
ENDSPLINE
```

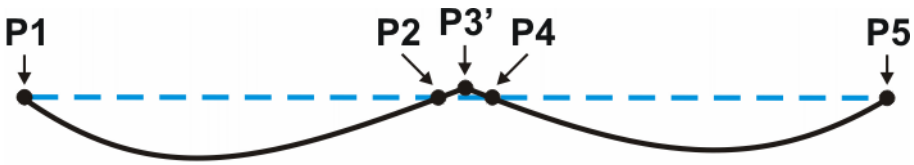


Fig. 9-26: Point has been offset

Remedy:

- Distribute the points more evenly
- Program straight lines (except very short ones) as SLIN segments

9.7.5 Approximation of spline motions

All spline blocks and all individual spline motions can be approximated with one another. It makes no difference whether they are CP or PTP spline blocks, nor is the motion type of the individual motion relevant.

The motion type of the approximate positioning arc always corresponds to the second motion. In the case of SPTP-SLIN approximation, for example, the approximate positioning arc is of type CP.

Spline motions cannot be approximated with conventional motions (LIN, CIRC, PTP).

Approximation not possible due to time or advance run stops:

If approximation is not possible for reasons of time or due to an advance run stop, the robot waits at the start of the approximate positioning arc.

- In the case of time reasons: the robot moves again as soon as it has been possible to plan the next block.
- In the case of an advance run stop: the end of the current block is reached at the start of the approximate positioning arc. This means that the advance run stop is canceled and the robot controller can plan the next block. Robot motion is resumed.

In both cases, the robot now moves along the approximate positioning arc. Approximate positioning is thus technically possible; it is merely delayed.

This response differs from that for LIN, CIRC or PTP motions. If approximate positioning is not possible for the reasons specified, the motion is executed to the end point with exact positioning.

No approximate positioning in MSTEP and ISTEP:

In the program run modes MSTEP and ISTEP, the robot stops exactly at the end point, even in the case of approximated motions.

In the case of approximate positioning from one spline block to another spline block, the result of this exact positioning is that the path is different in the last segment of the first block and in the first segment of the second block from the path in program run mode GO.

In all other segments of both spline blocks, the path is identical in MSTEP, ISTEP and GO.

9.7.6 Replacing an approximated CP motion with a spline block

Description

In order to replace approximated conventional CP motions with spline blocks, the program must be modified as follows:

- Replace LIN - LIN with SLIN - SPL - SLIN.
- Replace LIN - CIRC with SLIN - SPL - SCIRC.

  Recommendation: Allow the SPL to project a certain way into the original circle. The SCIRC thus starts later than the original CIRC.

In approximated motions, the corner point is programmed. In the spline block, the points at the start and end of the approximation are programmed instead.

The following approximated motion is to be reproduced:

```
LIN P1 C_DIS
LIN P2
```

Spline motion:

```
SPLINE
SLIN P1A
SPL P1B
SLIN P2
ENDSPLINE
```

P1A = start of approximation, P1B = end of approximation

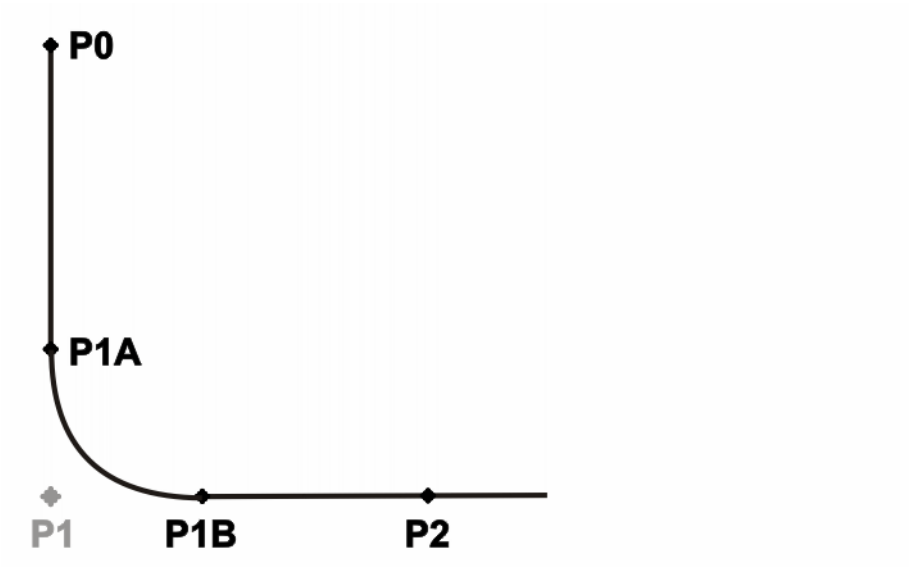


Fig. 9-27: Approximated motion - spline motion

Ways of determining P1A and P1B:

- Execute the approximated path and save the positions at the desired point by means of Trigger.
- Calculate the points in the program with KRL.
- The start of the approximation can be determined from the approximate positioning criterion. Example: If C_DIS is specified as the approximate positioning criterion, the distance from the start of the approximation to the corner point corresponds to the value of $APO.CDIS.

  The end of the approximation is dependent on the programmed velocity.

The SPL path does not correspond exactly to the approximate positioning arc, even if P1A and P1B are exactly at the start/end of the approximation. In order to recreate the exact approximate positioning arc, additional points must be inserted into the spline. Generally, one point is sufficient.

Example

The following approximated motion is to be reproduced:

```
$APO.CDIS=20
$VEL.CP=0.5
LIN {Z 10} C_DIS
LIN {Y 60}
```

Spline motion:

```
SPLINE WITH $VEL.CP=0.5
SLIN {Z 30}
SPL {Y 30, Z 10}
SLIN {Y 60}
ENDSPLINE
```

The start of the approximate positioning arc has been calculated from the approximate positioning criterion.

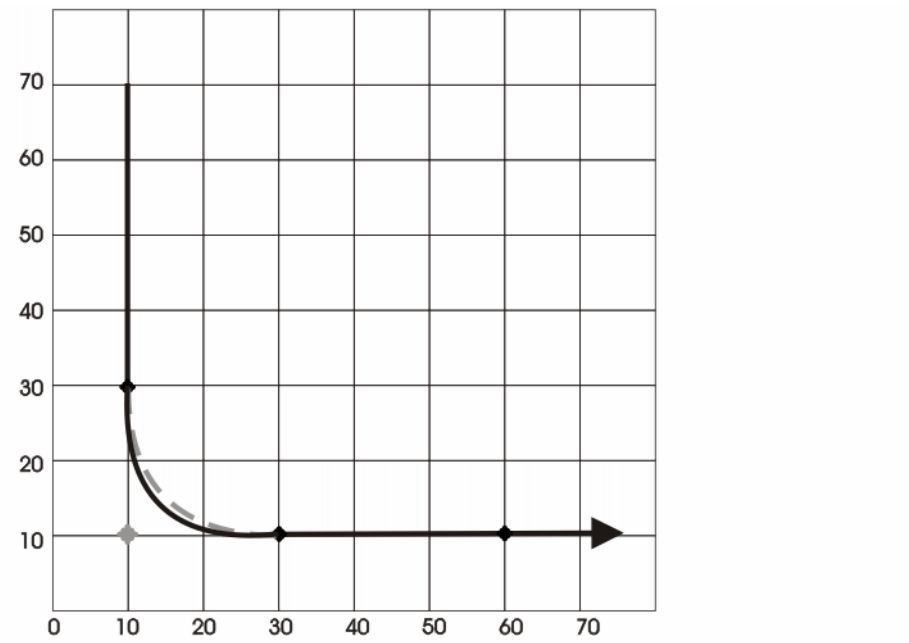


Fig. 9-28: Example: Approximated motion - spline motion 1

The SPL path does not yet correspond exactly to the approximate positioning arc. For this reason, an additional SPL segment is inserted into the spline.

```
SPLINE WITH $VEL.CP=0.5
SLIN {Z 30}
SPL {Y 15, Z 15}
SPL {Y 30, Z 10}
SLIN {Y 60}
ENDSPLINE
```

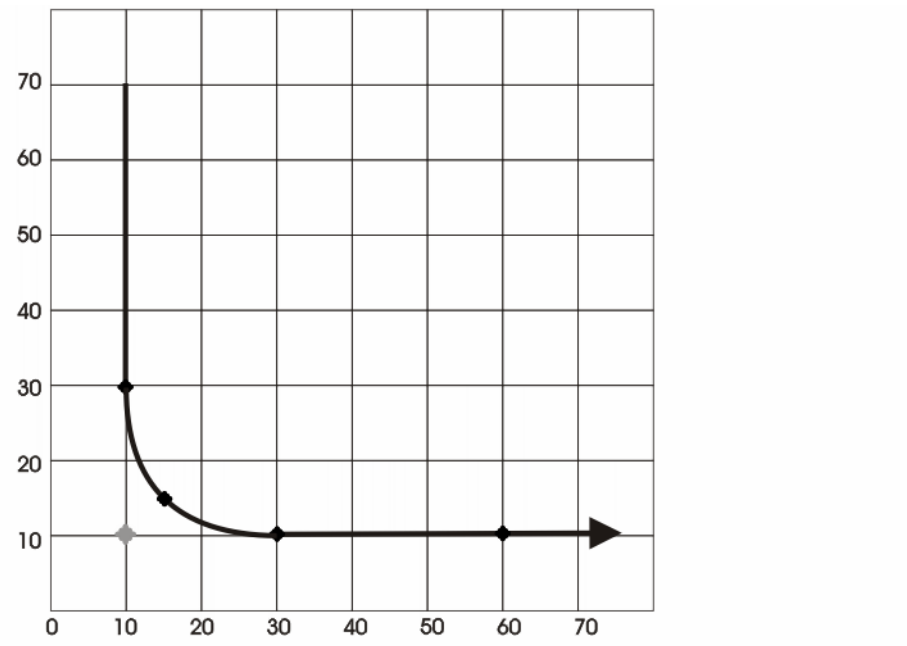


Fig. 9-29: Example: Approximated motion - spline motion 2

With the additional point, the path now corresponds to the approximate positioning arc.

9.7.6.1 SLIN-SPL-SLIN transition

In the case of a SLIN-SPL-SLIN segment sequence, it is usually desirable for the SPL segment to be located within the smaller angle between the two straight lines. Depending on the start and end point of the SPL segment, the path may also move outside this area.

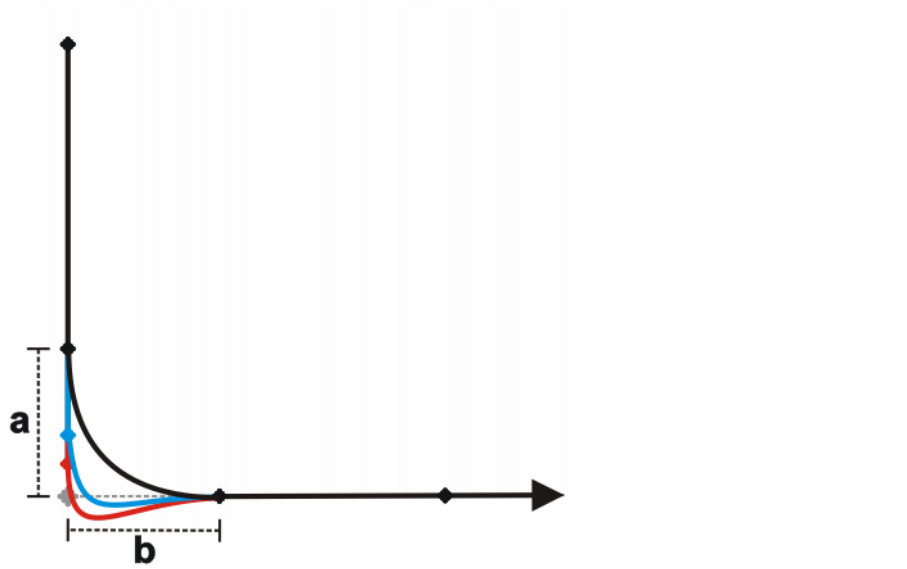


Fig. 9-30: SLIN-SPL-SLIN

The path remains inside if the following conditions are met:

- The extensions of the two SLIN segments intersect.

- 2/3 ≤ a/b ≤ 3/2

  a = distance from start point of the SPL segment to intersection of the SLIN segments

  b = distance from intersection of the SLIN segments to end point of the SPL segment

9.8 Orientation control for CP spline motions

Description

The orientation of the TCP can be different at the start point and end point of a motion. When a CP spline motion is programmed, it is necessary to select how to deal with the different orientations.

The orientation control type is defined as follows:

- If programming with KRL syntax: by means of the system variable $ORI_TYPE
- If programming with inline forms: in the option window **Motion parameters**

Orientation control	Description
• Option window: **Constant orientation** • $ORI_TYPE = #CONSTANT	The orientation of the TCP remains constant during the motion. The orientation of the start point is retained. The programmed orientation of the end point is not taken into consideration.
• Option window: **Default** • $ORI_TYPE = #VAR	The orientation of the TCP changes continuously during the motion. At the end point, the TCP has the programmed orientation.
• Option window: **Wrist PTP** • $ORI_TYPE = #JOINT	The orientation of the TCP changes continuously during the motion. This is done by linear transformation (axis-specific motion) of the wrist axis angles. **Note**: Use **Wrist PTP** if, with **Default**, the robot passes through a wrist axis singularity. The orientation of the TCP changes continuously during the motion, but not uniformly. **Wrist PTP** is thus not suitable if a specific orientation must be maintained exactly, e.g. in the case of laser welding.
• Option window: **Ignore orientation** • $ORI_TYPE = #IGNORE	This option is only available for CP spline segments (not for the spline block or for individual spline motions). This option is used if no specific orientation is required at a specific point. (>>> *"#IGNORE" Page 377*)

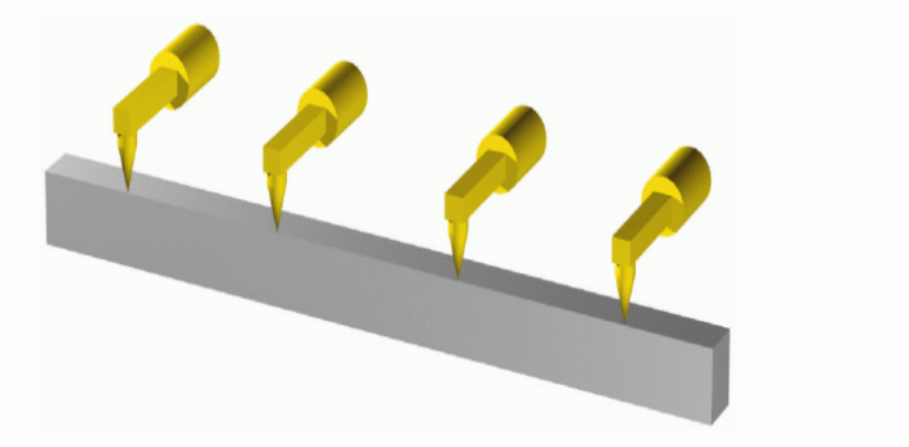


Fig. 9-31: Constant orientation

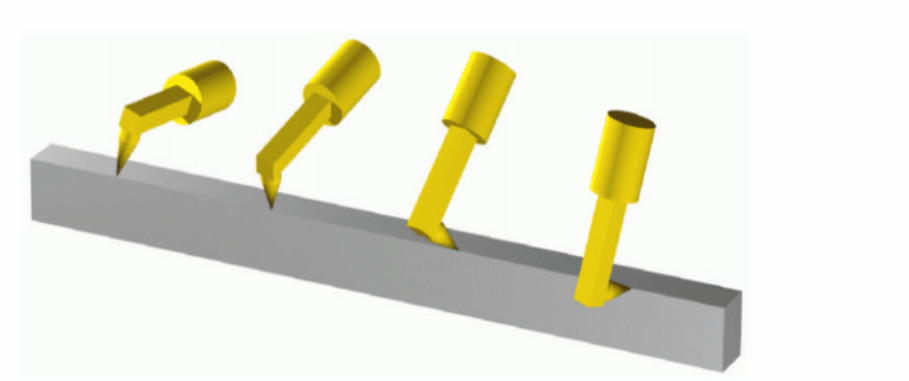


Fig. 9-32: Default orientation control

#IGNORE

$ORI_TYPE = #IGNORE is used if no specific orientation is required at a point. If this option is selected, the robot controller ignores the taught or programmed orientation of the point. Instead, it calculates the optimal orientation for this point on the basis of the orientations of the surrounding points. This reduces the cycle time.

Example:

```
SPLINE
SPL XP1
SPL XP2
SPL XP3 WITH $ORI_TYPE=#IGNORE
SPL XP4 WITH $ORI_TYPE=#IGNORE
SPL XP5
SPL XP6
ENDSPLINE
```

The taught or programmed orientation of XP3 and XP4 is ignored.

Characteristics of $ORI_TYPE = #IGNORE:

- In the program run modes MSTEP and ISTEP, the robot stops with the orientations calculated by the robot controller.
- In the case of a block selection to a point with #IGNORE, the robot adopts the orientation calculated by the robot controller.

$ORI_TYPE = #IGNORE is not allowed for the following segments:

- The last segment in a spline block

- SCIRC segments with $CIRC_TYPE=#PATH
- Segments followed by a SCIRC segment with $CIRC_TYPE=#PATH
- Segments followed by a segment with $ORI_TYPE=#CONSTANT

SCIRC

It is possible to define for SCIRC motions whether the orientation control is to be space-related or path-related.

(>>> *9.8.1 "SCIRC: reference system for the orientation control" Page 378*)

It is possible to define for SCIRC motions whether, and to what extent, the orientation of the auxiliary point is to be taken into consideration. The orientation behavior at the end point can also be defined.

(>>> *9.8.2 "SCIRC: orientation behavior" Page 379*)

$SPL_ORI_JOINT_AUTO

$SPL_ORI_JOINT_AUTO is used to optimize the motion characteristics in the vicinity of wrist axis singularities.

Functional principle of $SPL_ORI_JOINT_AUTO = #ON:

- For CP spline motions with $ORI_TYPE = #VAR, the robot controller automatically decides for each motion (i.e. for each segment) whether to execute it as #VAR or #JOINT.

$SPL_ORI_JOINT_AUTO = #ON is an alternative to using $ORI_TYPE = #JOINT. While $ORI_TYPE = #JOINT can be used for specific individual motions, $SPL_ORI_JOINT_AUTO = #ON enables automatic optimization over program sequences of any size with minimal effort for modification.

$SPL_ORI_JOINT_AUTO can only be modified using a robot program. $SPL_ORI_JOINT_AUTO cannot be used in spline segments.

Default: $SPL_ORI_JOINT_AUTO = #OFF

9.8.1 SCIRC: reference system for the orientation control

It is possible to define for SCIRC motions whether the orientation control is to be space-related or path-related. This can be defined as follows:

- If programming with inline forms: in the option window **Motion parameters**
- If programming with KRL syntax: by means of the system variable $CIRC_TYPE

Orientation control	Description
• Option window: **base-related** • $CIRC_TYPE = #BASE	Base-related orientation control during the circular motion
• Option window: **path-oriented** • $CIRC_TYPE = #PATH	Path-related orientation control during the circular motion

$CIRC_TYPE = #PATH is not allowed for the following motions:

- SCIRC segments for which $ORI_TYPE = #IGNORE
- SCIRC motions preceded by a spline segment for which $ORI_TYPE = #IGNORE

(>>> *9.6.1 "Combinations of $ORI_TYPE and $CIRC_TYPE" Page 359*)

9.8.2 SCIRC: orientation behavior

Description

During SCIRC motions, the robot controller can take the programmed orientation of the auxiliary point into consideration. The operator can define whether and to what extent it is actually taken into consideration:

- If programming with KRL syntax: by means of the system variable $CIRC_MODE
- If programming with inline forms: in the option window **Motion parameters**, tab **Circle configuration**

In the case of SCIRC statements with circular angles, the same procedure can also be used to define whether the end point is to have the programmed orientation or whether the orientation is to be scaled according to the circular angle.

$CIRC_MODE can only be written to by means of a SCIRC statement. $CIRC_MODE cannot be read.

Syntax

For auxiliary points:

`$CIRC_MODE.AUX_PT.ORI =` *BehaviorAUX*

For end points:

`$CIRC_MODE.TARGET_PT.ORI =` *BehaviorEND*

Explanation of the syntax

Element	Description
Behavior-AUX	Data type: ENUM • **#INTERPOLATE**: At the auxiliary point, the TCP adopts the programmed orientation. • **#IGNORE**: The robot controller ignores the programmed orientation of the auxiliary point. The transition from the start orientation of the TCP to the end orientation is carried out over the shortest possible distance. • **#CONSIDER** (default): There are essentially 2 paths for the transition from the start orientation to the end orientation by means of a rotation: a shorter one and a longer one. With #CONSIDER, the robot controller selects the path that comes closest to the programmed orientation of the auxiliary point. It is possible for the TCP to adopt the programmed orientation of the auxiliary point at some point along the path. This is not necessarily the case, however.
Behavior-END	Data type: ENUM • **#INTERPOLATE**: The programmed orientation of the end point is accepted at the actual end point. (Only possibility for SCIRC without specification of circular angle. If #EXTRAPOLATE is set, #INTERPOLATE is nonetheless executed.) • **#EXTRAPOLATE**: The orientation is adapted to the circular angle: If the circular angle makes the motion longer, the programmed orientation is accepted at the programmed end point. The orientation is continued accordingly to the actual end point. If the circular angle makes the motion shorter, the programmed orientation is not reached. (Default for SCIRC with specification of circular angle.)

Limitations

- If $ORI_TYPE = #IGNORE for a SCIRC segment, $CIRC_MODE is not evaluated.
- If a SCIRC segment is preceded by a SCIRC or SLIN segment with $ORI_TYPE = #IGNORE, #CONSIDER cannot be used in this SCIRC segment.

For SCIRC with circular angle:

- #INTERPOLATE must not be set for the auxiliary point.
- If $ORI_TYPE = #IGNORE, #EXTRAPOLATE must not be set for the end point.
- If it is preceded by a spline segment with $ORI_TYPE = #IGNORE, #EXTRAPOLATE must not be set for the end point.

9.8.2.1 SCIRC: Orientation behavior – example: auxiliary point

Description

The following orientations have been programmed for the TCP:

- Start point: 0°
- Auxiliary point: 98°
- End point: 197°

The re-orientation is thus 197°. If the auxiliary point is ignored, the end orientation can also be achieved by means of the shorter re-orientation of 360° - 197° = 163°.

- The dotted orange arrows show the programmed orientation.
- The gray arrows indicate schematically what the actual orientation would be where this differs from the programmed orientation.

#INTERPOLATE

At the auxiliary point, the TCP adopts the programmed orientation of 98°. The re-orientation is 197°.

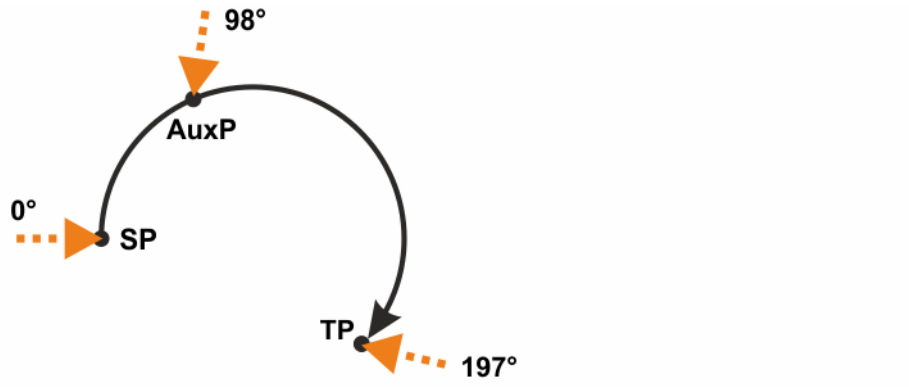


Fig. 9-33: #INTERPOLATE

SP	Start point
AuxP	Auxiliary point
TP	End point

#IGNORE

The short re-orientation through 163° is used. The programmed orientation of the auxiliary point is disregarded.

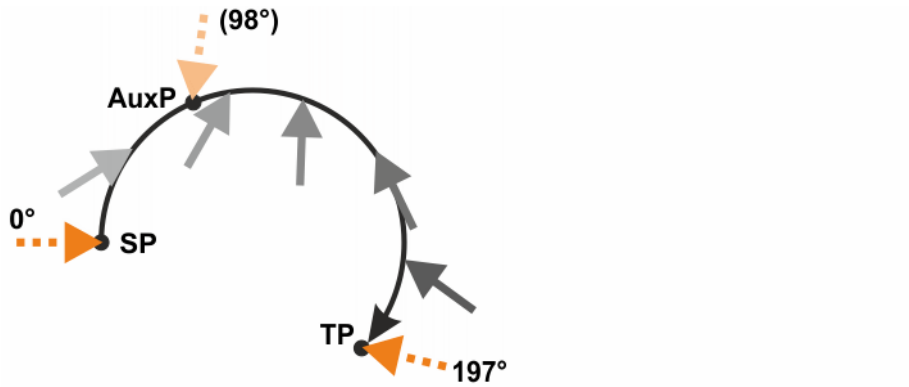


Fig. 9-34: #IGNORE

#CONSIDER

> #CONSIDER is suitable if the user wants to specify the re-orientation direction of the TCP without the need for a specific orientation at the auxiliary point. The user can specify the direction using the auxiliary point.

The programmed orientation of the auxiliary point is 98° and is thus on the longer path. The robot controller thus selects the longer path for the re-orientation.

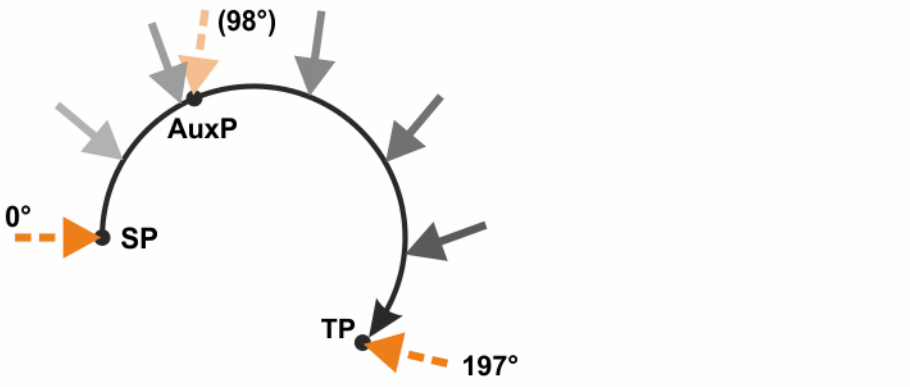


Fig. 9-35: #CONSIDER

Additional example for #CONSIDER:

If the auxiliary point were to be programmed with 262°, it would be on the shorter path. The robot controller would therefore select the shorter path for the re-orientation. The gray arrows indicate that it does not necessarily adopt the programmed orientation of the auxiliary point.

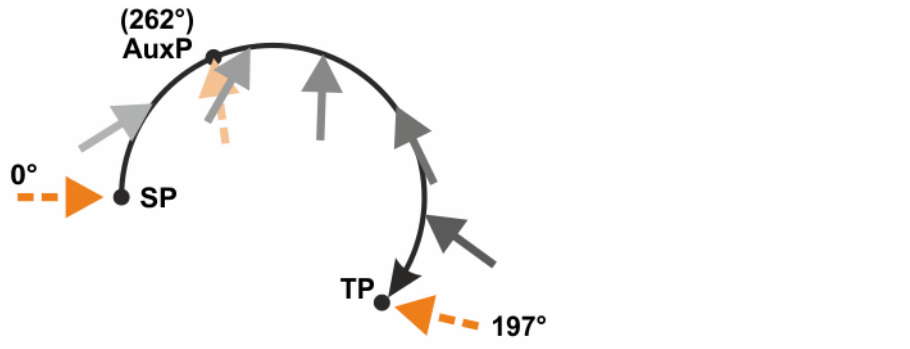


Fig. 9-36: #CONSIDER, additional example

9.8.2.2 SCIRC: Orientation behavior – example: end point

Description

- The dotted orange arrows show the programmed orientation.
- The gray arrows show the actual orientation where this differs from the programmed orientation.

#INTERPOLATE

At **TP**, which is situated before **TP_CA**, the programmed orientation has not yet been reached. The programmed orientation is accepted at **TP_CA**.

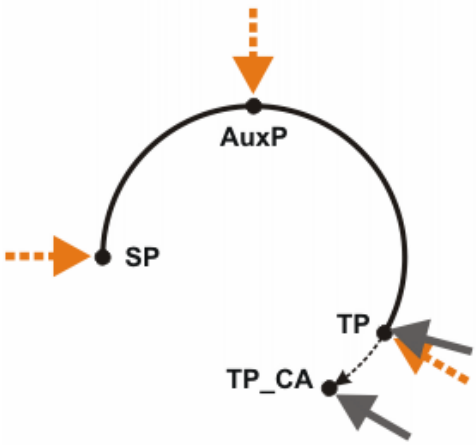


Fig. 9-37: #INTERPOLATE

SP	Start point
AuxP	Auxiliary point
TP	Programmed end point
TP_CA	Actual end point. Determined by the circular angle.

#EXTRAPOLATE

The programmed orientation is accepted at **TP**. For **TP_CA**, this orientation is continued in accordance with the circular angle.

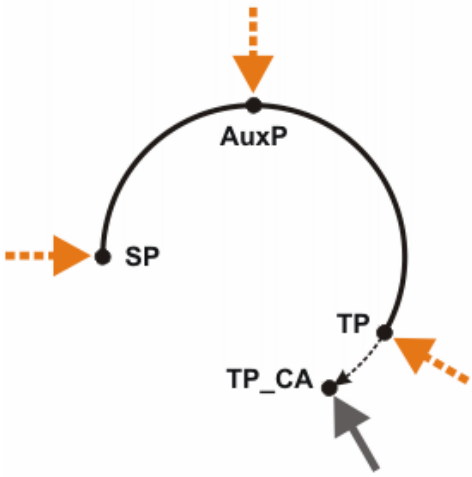


Fig. 9-38: #EXTRAPOLATE

9.9 Circular angle

A circular angle can be programmed for most circular motions.

> Information about whether this is possible for a specific circular motion can be found in the description of the motion in the programming section of this documentation.

The circular angle specifies the overall angle of the motion. This makes it possible to extend the motion beyond the programmed end point or to shorten it. The actual end point thus no longer corresponds to the programmed end point.

Unit: degrees. Circular angles greater than +360° and less than -360° can be programmed.

The preceding sign determines the direction in which the circular path is executed:

- Positive: direction **Start point › Auxiliary point › End point**
- Negative: direction **Start point › End point › Auxiliary point**

9.10 Status and Turn

Overview

The position (X, Y, Z) and orientation (A, B, C) of the TCP are not sufficient to define the axis angles of a robot unambiguously. Status and Turn are additionally required for this.

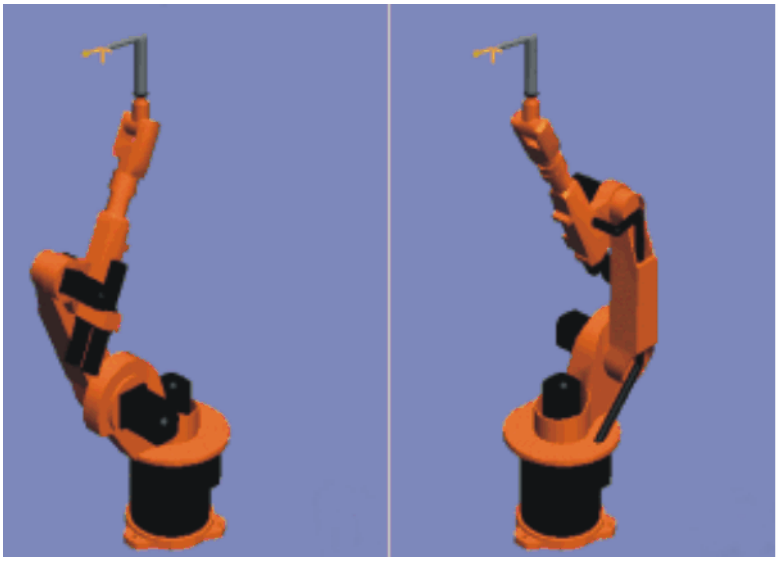


Fig. 9-39: Example: Same TCP position, different axis position

Status (S) and Turn (T) are integral parts of the data types POS and E6POS:

```
STRUC POS REAL X, Y, Z, A, B, C, INT S, T
```

```
STRUC E6POS REAL X, Y, Z, A, B, C, E1, E2, E3, E4, E5, E6,
INT S, T
```

KRL program

The robot controller only takes the programmed Status and Turn values into consideration for PTP motions. They are ignored for CP motions.

The first motion instruction in a KRL program must therefore be one of the following instructions so that an unambiguous starting position is defined for the robot:

- A complete PTP instruction of type POS or E6POS
- Or a complete PTP instruction of type AXIS or E6AXIS

“Complete” means that all components of the end point must be specified. The default HOME position is always a complete PTP instruction.

Status and Turn can be omitted in the subsequent instructions:

- The robot controller retains the previous Status value.

- The Turn value is determined by the path in CP motions. In the case of PTP motions, the robot controller selects the Turn value that results in the shortest possible path.

9.10.1 Status

The Status consists of a sequence of bits which, together with the position of the TCP, defines the axis position of the robot.

Bit 0

Bit 0 specifies the position of the intersection of the wrist axes (A4, A5, A6).

Position	Value
Overhead area If the x-value of the intersection of the wrist axes, relative to the A1 coordinate system, is negative, the robot is in the overhead area.	Bit 0 = 1
Basic area If the x-value of the intersection of the wrist axes, relative to the A1 coordinate system, is positive, the robot is in the basic area.	Bit 0 = 0

The A1 coordinate system is identical to the $ROBROOT coordinate system if axis 1 is at 0°. For values not equal to 0°, it moves with axis 1.

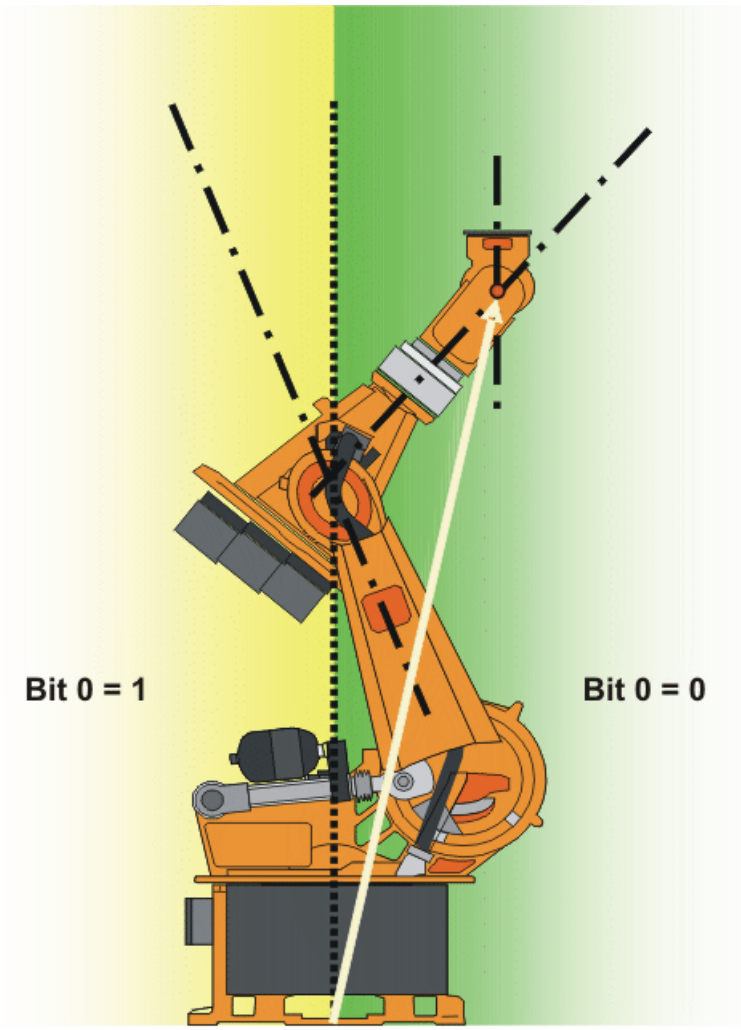


Fig. 9-40: Example: The intersection of the wrist axes (red dot) is in the basic area.

Bit 1

Bit 1 specifies the position of axis A3.

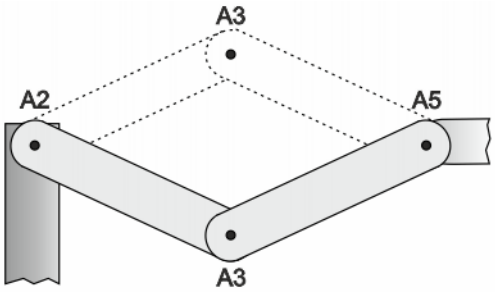


Fig. 9-41: Bit 1 – position of A3

The angle at which the value of bit 1 changes depends on the robot type. For a KR 30 or KR 60, the offset between axes A3 and A4 must be taken into consideration.

For robots whose axes A3 and A4 intersect (no offset between the axes), the following applies:

Position	Value
A3 ≥ 0°	Bit 1 = 1
A3 < 0°	Bit 1 = 0

For robots with an offset between axes A3 and A4, the angle at which the value of bit 1 changes depends on the size of this offset.

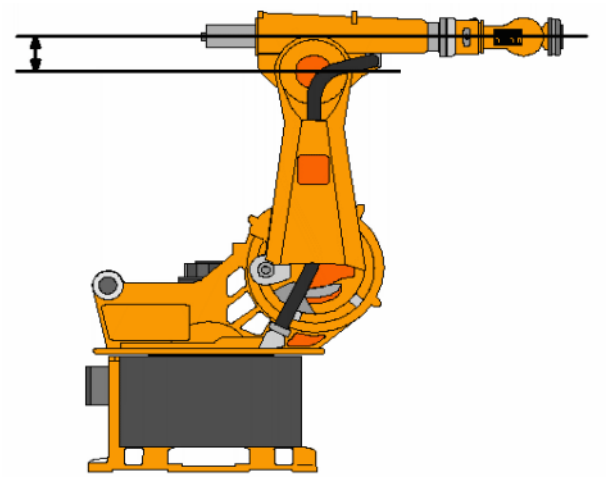


Fig. 9-42: Offset between A3 and A4: example KR 30

Bit 2

Bit 2 specifies the position of axis A5.

Position (reference: A4 = 0°)	Value
A5 is tilted upwards.	Bit 2 = 1
A5 = 0°	Bit 2 = 0
A5 is tilted downwards.	

The sign preceding the angle of A5 is irrelevant! What counts is the direction in which A5 is tilted, i.e. upwards or downwards. The direction is specified with reference to the zero position of A4.

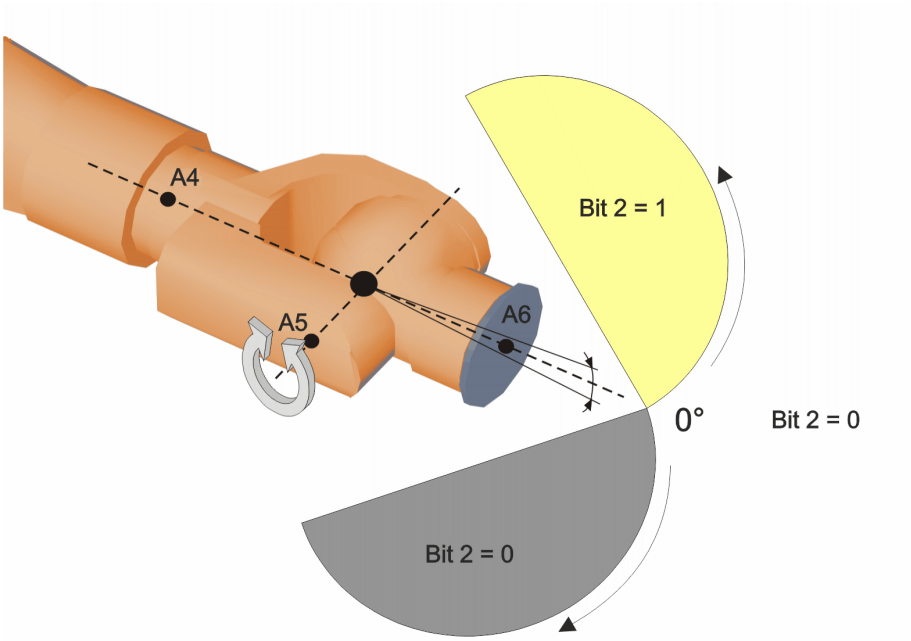


Fig. 9-43: Bit 2

Bit 3

Bit 3 is not used and is always 0.

Bit 4

Bit 4 specifies whether or not the point was taught using an absolutely accurate robot.

Depending on the value of the bit, the point can be executed by both absolutely accurate robots and non-absolutely-accurate robots. Bit 4 is for information purposes only and has no influence on how the robot calculates the point. This means, therefore, that when a robot is programmed offline, bit 4 can be ignored.

Description	Value
The point was not taught with an absolutely accurate robot.	Bit 4 = 0
The point was taught with an absolutely accurate robot.	Bit 4 = 1

9.10.2 Turn

Description

Turn specifies the sign preceding the axis angles.

The Turn specification makes it possible to move axes through angles greater than +180° or less than -180° without the need for special motion strategies (e.g. auxiliary points). With rotational axes, the individual bits determine the sign before the axis angle in the following way:

Bit = 0: Angle ≥ 0°

Bit = 1: Angle < 0°

Value	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	A6 ≥ 0°	A5 ≥ 0°	A4 ≥ 0°	A3 ≥ 0°	A2 ≥ 0°	A1 ≥ 0°
1	A6 < 0°	A5 < 0°	A4 < 0°	A3 < 0°	A2 < 0°	A1 < 0°

Example

```
DECL POS XP1 = {X 900, Y 0, Z 800, A 0, B 0, C 0, S 6, T
19}
```

T 19 corresponds to T 'B010011'. This means:

Axis	Angle
A1	negative
A2	negative
A3	positive
A4	positive
A5	negative
A6	positive

9.11 Singularities

KUKA robots with 6 degrees of freedom have 3 different singularity positions.

- Overhead singularity
- Extended position singularity
- Wrist axis singularity

A singularity position is characterized by the fact that unambiguous reverse transformation (conversion of Cartesian coordinates to axis-specific values) is not possible, even though Status and Turn are specified. In this case, or if very slight Cartesian changes cause very large changes to the axis angles, one speaks of singularity positions.

Overhead

In the overhead singularity, the wrist root point (intersection of axes A4, A5 and A6) is located vertically above axis 1.

The position of axis A1 cannot be determined unambiguously by means of reverse transformation and can thus take any value.

If the end point of a PTP motion is situated in this overhead singularity position, the robot controller may react as follows by means of the system variable $SINGUL_POS[1]:

- **0**: The angle for axis A1 is defined as 0 degrees (default setting).
- **1**: The angle for axis A1 remains the same from the start point to the end point.

Extended position

In the extended position singularity, the wrist root point (intersection of axes A4, A5 and A6) is located in the extension of axes A2 and A3 of the robot.

The robot is at the limit of its work envelope.

Although reverse transformation does provide unambiguous axis angles, low Cartesian velocities result in high axis velocities for axes A2 and A3.

If the end point of a PTP motion is situated in this extended position singularity, the robot controller may react as follows by means of the system variable $SINGUL_POS[2]:

- **0**: The angle for axis A2 is defined as 0 degrees (default setting).
- **1**: The angle for axis A2 remains the same from the start point to the end point.

Wrist axes

In the wrist axis singularity position, the axes A4 and A6 are parallel to one another and axis A5 is within the range ±0.01812°.

The position of the two axes cannot be determined unambiguously by reverse transformation. There is an infinite number of possible axis positions for axes A4 and A6 with identical axis angle sums.

If the end point of a PTP motion is situated in this wrist axis singularity, the robot controller may react as follows by means of the system variable $SINGUL_POS[3]:

- **0**: The angle for axis A4 is defined as 0 degrees (default setting).
- **1**: The angle for axis A4 remains the same from the start point to the end point.

10 Programming with inline forms

10.1 Instructions for programming

NOTICE

When programming motions, it must be ensured that the energy supply system is not wound up or damaged during program execution.

NOTICE

In the case of programs with the following axis motions or positions, the film of lubricant on the gear units of the axes may break down:

- Motions <3°
- Oscillating motions
- Areas of gear units permanently facing upwards

It must be ensured that the gear units have a sufficient supply of oil. For this, in the case of oscillating motions or short motions (<3°), programming must be carried out in such a way that the affected axes regularly move more than 40° (e.g. once per cycle).
In the case of areas of gear units permanently facing upwards, sufficient oil supply must be achieved by programming re-orientations of the inline wrist. In this way, the oil can reach all areas of the gear units by means of gravity. Required frequency of re-orientations:

- With low loads (gear unit temperature <+35 °C): daily
- With medium loads (gear unit temperature +35 °C to 55 °C): hourly
- With heavy loads (gear unit temperature >+55 °C): every 10 minutes

Failure to observe this precaution may result in damage to the gear units.

10.2 Names in inline forms

Names for data sets can be entered in inline forms. These include, for example, point names, names for motion data sets, etc.

The following restrictions apply to names:

- Maximum length 23 characters; 22 characters for names of global points
- No special characters are permissible, with the exception of $.
- The first character must not be a number.

The restrictions do not apply to output names.

Other restrictions may apply in the case of inline forms in technology packages.

10.3 Programming PTP, LIN and CIRC motions

10.3.1 Programming a PTP motion

Precondition

- User rights: Function group **Old motion range inline forms**
- A program is selected.
- T1 mode

Procedure

1. Move the TCP to the position that is to be taught as the end point.
2. Position the cursor in the line after which the motion instruction is to be inserted.
3. Select the menu sequence **Commands** > **Motion** > **PTP**.
4. Set the parameters in the inline form.
   (>>> *10.3.2 "Inline form “PTP”" Page 392*)
5. Save instruction with **Cmd OK**.

10.3.2 Inline form “PTP”

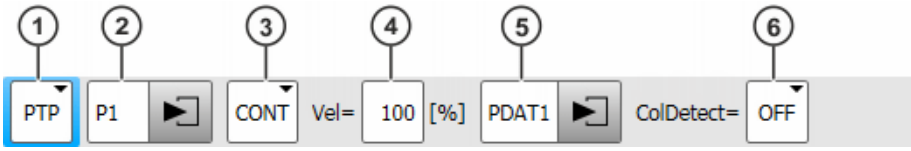


Fig. 10-1: Inline form for PTP motions

Item	Description
1	Motion type **PTP**
2	Name of the end point. The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the point data. The corresponding option window is opened. (>>> *10.3.7 "Option window “Frames”" Page 395*) The arrow can also be used to edit the **Global point** setting.
3	• **CONT**: end point is approximated. • **[Empty box]**: The motion stops exactly at the end point.
4	Velocity • **1 … 100%**
5	Name for the motion data set The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the point data. The corresponding option window is opened. (>>> *10.3.8 "Option window “Motion parameters” (LIN, CIRC, PTP)" Page 396*)
6	Collision detection for this motion • **OFF**: The collision detection is switched off. • **CDSet_Set[***No.***]**: The collision detection is switched on. The values from data set *No.* are used for detection. (>>> *8.14.6 "Activating/deactivating collision detection via inline form" Page 348*)

10.3.3 Programming a LIN motion

Precondition

- User rights: Function group **Old motion range inline forms**
- A program is selected.
- T1 mode

Procedure

1. Move the TCP to the position that is to be taught as the end point.
2. Position the cursor in the line after which the motion instruction is to be inserted.
3. Select the menu sequence **Commands** > **Motion** > **LIN**.
4. Set the parameters in the inline form.
   (>>> *10.3.4 "Inline form “LIN”" Page 393*)
5. Save instruction with **Cmd OK**.

10.3.4 Inline form “LIN”

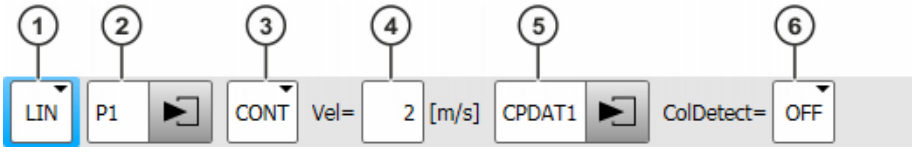


Fig. 10-2: Inline form for LIN motions

Item	Description
1	Motion type **LIN**
2	Name of the end point. The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the point data. The corresponding option window is opened. (>>> *10.3.7 "Option window “Frames”" Page 395*) The arrow can also be used to edit the **Global point** setting.
3	• **CONT**: end point is approximated. • **[Empty box]**: The motion stops exactly at the end point.
4	Velocity • **0.001 … 2 m/s**
5	Name for the motion data set The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the point data. The corresponding option window is opened. (>>> *10.3.8 "Option window “Motion parameters” (LIN, CIRC, PTP)" Page 396*)

Item	Description
6	Collision detection for this motion • **OFF**: The collision detection is switched off. • **CDSet_Set[***No.***]**: The collision detection is switched on. The values from data set *No.* are used for detection. (>>> *8.14.6 "Activating/deactivating collision detection via inline form" Page 348*)

10.3.5 Programming a CIRC motion

Precondition

- User rights: Function group **Old motion range inline forms**
- A program is selected.
- T1 mode

Procedure

1. Move the TCP to the position that is to be taught as the auxiliary point.
2. Position the cursor in the line after which the motion instruction is to be inserted.
3. Select the menu sequence **Commands** > **Motion** > **CIRC**.
4. Set the parameters in the inline form.
   (>>> *10.3.6 "Inline form “CIRC”" Page 394*)
5. Press **Teach Aux**.
6. Move the TCP to the position that is to be taught as the end point.
7. Save instruction with **Cmd OK**.

10.3.6 Inline form “CIRC”

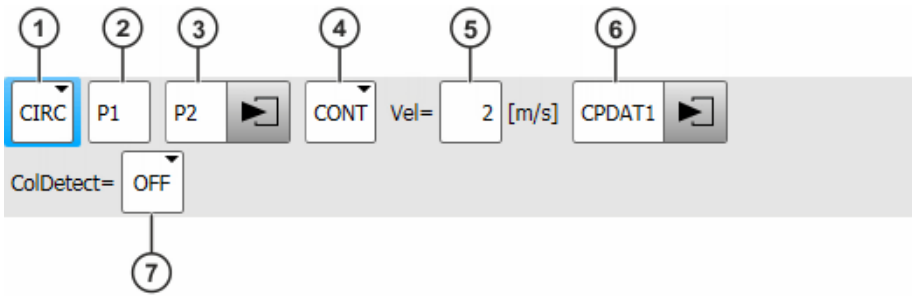


Fig. 10-3: Inline form for CIRC motions

Item	Description
1	Motion type **CIRC**
2	Name of the auxiliary point The system automatically generates a name. The name can be overwritten.

Item	Description
3	Name of the end point The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the point data. The corresponding option window is opened. (>>> *10.3.7 "Option window "Frames"" Page 395*)
4	• **CONT**: end point is approximated. • **[Empty box]**: The motion stops exactly at the end point.
5	Velocity • **0.001 … 2 m/s**
6	Name for the motion data set The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the point data. The corresponding option window is opened. (>>> *10.3.8 "Option window "Motion parameters" (LIN, CIRC, PTP)" Page 396*)
7	Collision detection for this motion • **OFF**: The collision detection is switched off. • **CDSet_Set[***No.***]**: The collision detection is switched on. The values from data set *No.* are used for detection. (>>> *8.14.6 "Activating/deactivating collision detection via inline form" Page 348*)

10.3.7 Option window "Frames"

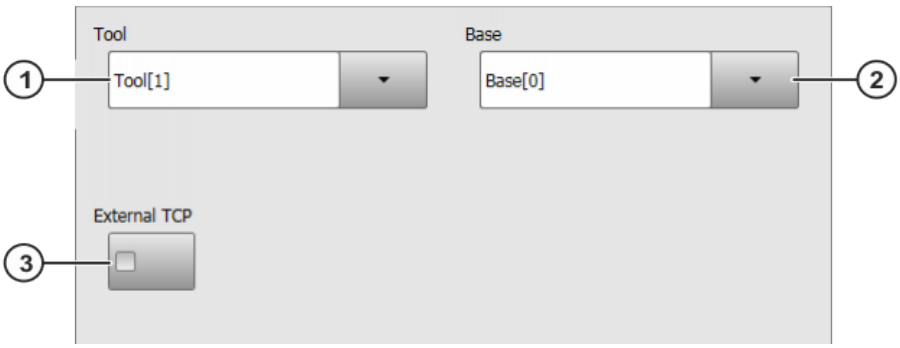


Fig. 10-4: Option window "Frames"

Item	Description
1	Select the tool or the workpiece on the flange. Range of values: TOOL[1] … TOOL[16]
2	Select the base or the fixed tool. Range of values: BASE[1] … BASE[32]

Item	Description
3	Specify the interpolation mode. • FALSE (check box not active): Set if the following combination has been selected above: – **Tool**: A tool on the flange – **Base**: A base • TRUE (check box active): Set if the following combination has been selected above: – **Tool**: A workpiece on the flange – **Base**: A fixed tool

10.3.8 Option window “Motion parameters” (LIN, CIRC, PTP)

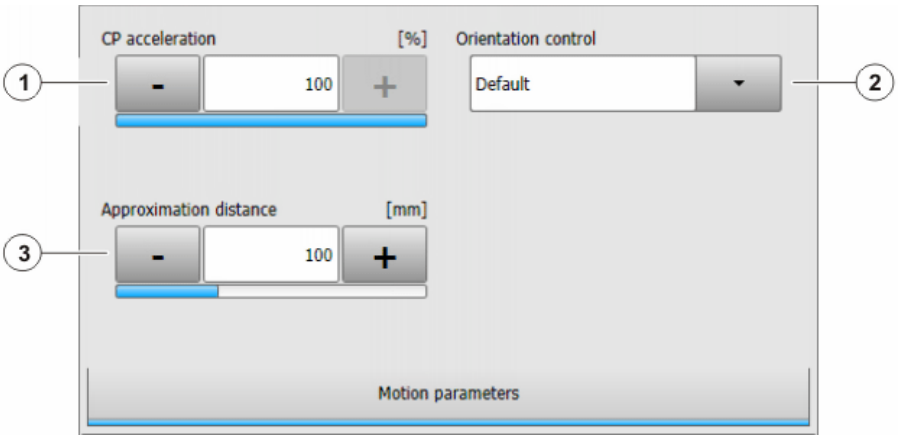


Fig. 10-5: Option window “Motion parameters” (LIN, CIRC, PTP)

Item	Description
1	Acceleration Refers to the maximum value specified in the machine data. The maximum value depends on the robot type and the selected operating mode.
2	This box is only displayed if it is specified in the inline form that the point is to be approximated. Furthest distance before the end point at which approximate positioning can begin The maximum permissible value is half the distance between the start point and the end point. If a higher value is entered, this is ignored and the maximum value is used.
3	This box is only displayed for LIN and CIRC motions. Orientation control selection. • **Standard** • **Wrist PTP** • **Constant orientation** (>>> *9.6 "Orientation control LIN, CIRC" Page 358*)

10.4 Programming spline motions

10.4.1 Programming tips for spline motions

- It is only possible to exploit the advantages of the spline motion type to the full if spline blocks are used.
- Interrupt programs must not contain any spline motions.
- A spline block should cover only one process (e.g. an adhesive seam). More than one process in a spline block leads to a loss of structural clarity within the program and makes changes more difficult.
- Use SLIN and SCIRC segments in cases where the workpiece necessitates straight lines and arcs. (Exception: use SPL segments for very short straight lines.) Otherwise, use SPL segments, particularly if the points are close together.
- Procedure for defining the path:
  1. First teach or calculate a few characteristic points. Example: points at which the curve changes direction.
  2. Test the path. At points where the accuracy is still insufficient, add more SPL points.
- Avoid successive SLIN and/or SCIRC segments, as this often reduces the velocity to 0.

  Program SPL segments between SLIN and SCIRC segments. The length of the SPL segments must be at least > 0.5 mm. Depending on the actual path, significantly larger SPL segments may be required.
- Avoid successive points with identical Cartesian coordinates, as this reduces the velocity to 0.
- The parameters (tool, base, velocity, etc.) assigned to the spline block have the same effect as assignments before the spline block. The assignment to the spline block has the advantage, however, that the correct parameters are read in the case of a block selection.
- Use the option **Ignore orientation** if no specific orientation is required for a SLIN, SCIRC or SPL segment. The robot controller calculates the optimal orientation for this point on the basis of the orientations of the surrounding points. This improves the cycle time.
- If there are external axes present and no specific position of the external axis is required for a spline segment, set $EX_AX_IGNORE to “1” for that external axis. The robot controller then calculates the optimal position for this point on the basis of the surrounding external axis positions. This improves the cycle time.
- The jerk can be modified The jerk is the change in acceleration. Procedure:
  1. Use the default values initially.
  2. If vibrations occur at tight corners: reduce values.

     If the velocity drops or the desired velocity cannot be reached: increase values or increase acceleration.
- If the robot executes points which lie on a work surface, a collision with the work surface is possible when approaching the first point.

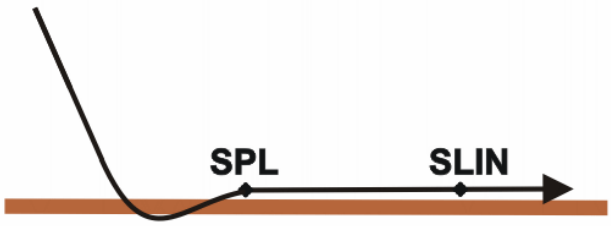


Fig. 10-6: Collision with work surface

In order to avoid a collision, observe the recommendations for the SLIN-SPL-SLIN transition.

(>>> *9.7.6.1 "SLIN-SPL-SLIN transition" Page 375*)

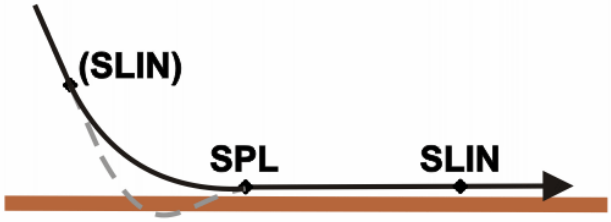


Fig. 10-7: Avoiding a collision with the work surface

- In the case of PTP spline blocks with multiple SPTP segments, it is possible that the software limit switches may be violated even though the points are within the limits!

  In this case, the points must be re-taught, i.e. they must be moved further away from the software limit switches. Alternatively, the software limit switches can be modified, provided that the required machine protection is still assured.

10.4.2 Programming a spline block

Description

A spline block can be used to group together several motions as an overall motion. The motions that may be included in a spline block are called spline segments. They are taught separately.

A spline block is planned and executed by the robot controller as a single motion block.

- A CP spline block may contain SPL, SLIN and SCIRC segments.
- A PTP spline block may contain SPTP segments.

A spline block that contains no segments is not a motion statement. The number of segments in the block is only limited by the memory capacity. Apart from the segments, a spline block may also contain the following elements:

- Inline commands from technology packages that support the spline functionality
- Comments and blank lines

A spline block must not include any other instructions, e.g. variable assignments or logic statements.

> The start point of a spline block is the last point before the spline block.
> The end point of a spline block is the last point in the spline block.
> A spline block does not trigger an advance run stop.

Precondition

- User rights: Function group **New motion range inline forms**
- A program is selected.
- T1 mode

Procedure

1. Position the cursor in the line after which the spline block is to be inserted.
2. Select the menu sequence **Commands** > **Motion**.
   - Then select **SPLINE block** for a CP spline block.
   - Or select **PTP SPLINE block** for a PTP spline block.
3. Set the parameters in the inline form.
   (>>> *10.4.2.1 "Inline form for CP spline block" Page 399*)
   (>>> *10.4.2.2 "Inline form “PTP SPLINE block”" Page 400*)
4. Press **Cmd OK**.
5. Press **Open/close fold**. Spline segments can now be inserted into the block.

10.4.2.1 Inline form for CP spline block

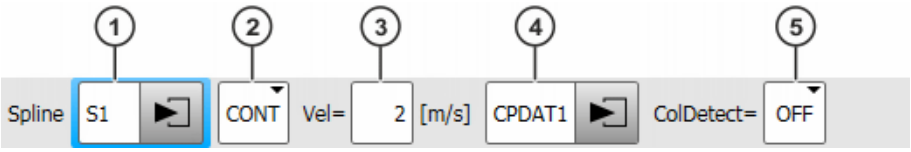


Fig. 10-8: Inline form for CP spline block

Item	Description
1	Name of the spline block. The system automatically generates a name. The name can be overwritten. Position the cursor in this box to edit the motion data. The corresponding option window is opened. (>>> *10.3.7 "Option window “Frames”" Page 395*)
2	• **CONT**: end point is approximated. • **[Empty box]**: The motion stops exactly at the end point.
3	Cartesian velocity • **0.001 … 2 m/s**
4	Name for the motion data set. The system automatically generates a name. The name can be overwritten. Position the cursor in this box to edit the motion data. The corresponding option window is opened. (>>> *10.4.2.3 "Option window “Motion parameters” (CP spline block)" Page 401*)

Item	Description
5	Collision detection for the spline block. The setting is valid for the segments for which the **ColDetect** box is not displayed. • **OFF**: The collision detection is switched off. • **CDSet_Set[***No.***]**: The collision detection is switched on. The values from data set *No.* are used for detection. (>>> *8.14.6 "Activating/deactivating collision detection via inline form" Page 348*)

10.4.2.2 Inline form “PTP SPLINE block”

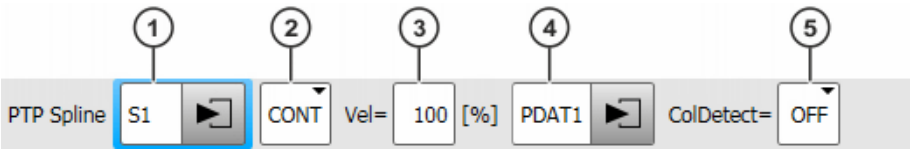


Fig. 10-9: Inline form “PTP SPLINE block”

Item	Description
1	Name of the spline block. The system automatically generates a name. The name can be overwritten. Position the cursor in this box to edit the motion data. The corresponding option window is opened. (>>> *10.3.7 "Option window “Frames”" Page 395*)
2	• **CONT**: end point is approximated. • **[Empty box]**: The motion stops exactly at the end point.
3	Axis velocity • **1 … 100%**
4	Name for the motion data set. The system automatically generates a name. The name can be overwritten. Position the cursor in this box to edit the motion data. The corresponding option window is opened. (>>> *10.4.2.4 "Option window “Motion parameters” (PTP spline block)" Page 402*)
5	Collision detection for the spline block. The setting is valid for the segments for which the **ColDetect** box is not displayed. • **OFF**: The collision detection is switched off. • **CDSet_Set[***No.***]**: The collision detection is switched on. The values from data set *No.* are used for detection. (>>> *8.14.6 "Activating/deactivating collision detection via inline form" Page 348*)

10.4.2.3 Option window “Motion parameters” (CP spline block)

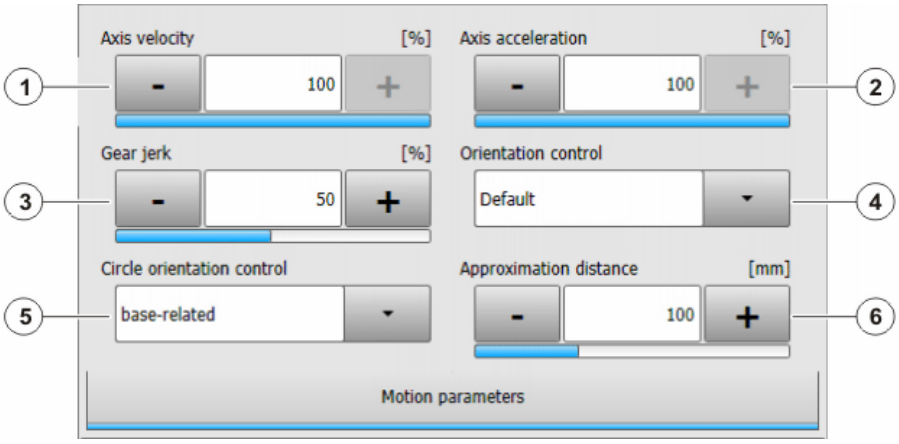


Fig. 10-10: Option window “Motion parameters” (CP spline block)

Item	Description
1	Axis velocity. The value refers to the maximum value specified in the machine data. • **1 … 100%**
2	Axis acceleration. The value refers to the maximum value specified in the machine data. • **1 … 100%**
3	Gear jerk. The jerk is the change in acceleration. The value refers to the maximum value specified in the machine data. • **1 … 100%**
4	Orientation control selection.
5	Orientation control reference system selection. This parameter only affects SCIRC segments (if present).
6	This box is only displayed if **CONT** was selected in the inline form. Furthest distance before the end point at which approximate positioning can begin. The maximum distance is that of the last segment in the spline. If there is only one segment present, the maximum distance is half the segment length. If a higher value is entered, this is ignored and the maximum value is used.

10.4.2.4 Option window “Motion parameters” (PTP spline block)

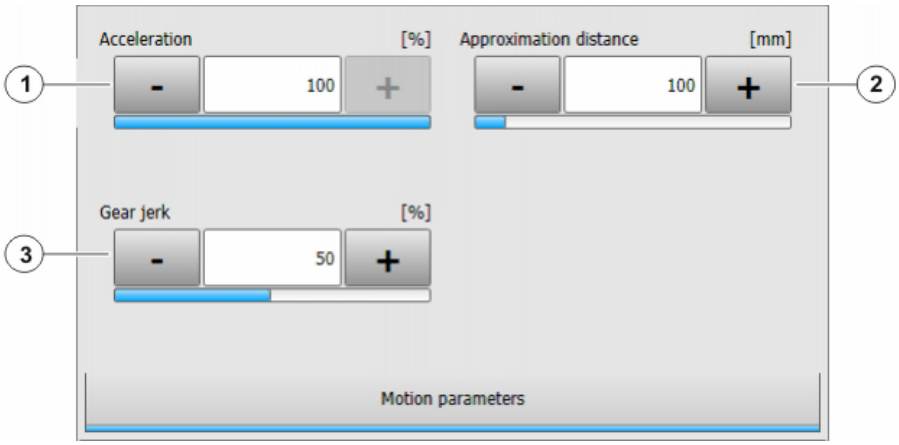


Fig. 10-11: Option window “Motion parameters” (PTP spline block)

Item	Description
1	Axis acceleration. The value refers to the maximum value specified in the machine data. • **1 … 100%**
2	This box is only displayed if **CONT** was selected in the inline form. Furthest distance before the end point at which approximate positioning can begin. The maximum distance is that of the last segment in the spline. If there is only one segment present, the maximum distance is half the segment length. If a higher value is entered, this is ignored and the maximum value is used.
3	Gear jerk. The jerk is the change in acceleration. The value refers to the maximum value specified in the machine data. • **1 … 100%**

10.4.3 Programming segments for a spline block

10.4.3.1 Programming an SPL or SLIN segment

Precondition

- User rights: Function group **New motion range inline forms**
- A program is selected.
- T1 mode
- The CP spline block fold is open.

Procedure

1. Move the TCP to the end point.
2. Position the cursor in the line after which the segment is to be inserted in the spline block.
3. Select the menu sequence **Commands** > **Motion** > **SPL** or **SLIN**.
4. Set the parameters in the inline form.

5. Press **Cmd OK**.

10.4.3.2 Inline form SPL spline segment/SLIN spline segment

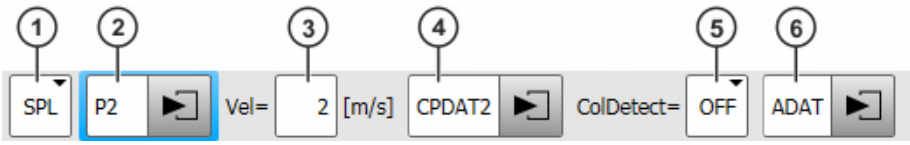


Fig. 10-12: Inline form SPL segment

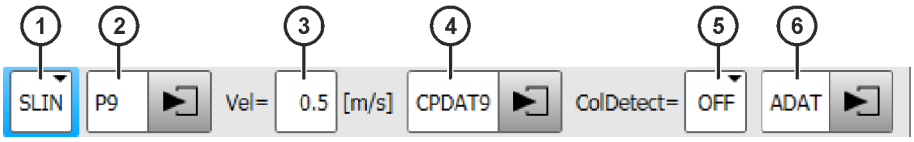


Fig. 10-13: Inline form SLIN segment

By default, not all boxes of the inline form are displayed. The boxes can be displayed or hidden using the **Switch parameter** button.

Item	Description
1	Motion type **SPL** or **SLIN**
2	Point name for the end point. The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the **Global point** setting. The corresponding window is opened.
3	Cartesian velocity By default, the value that is valid for the spline block is also valid for the segment. A separate value can be assigned here for the segment if required. The value applies only for this segment. • **0.001 … 2 m/s**
4	Name for the motion data set. The system automatically generates a name. The name can be overwritten. By default, the values that are valid for the spline block are also valid for the segment. Separate values can be assigned here for the segment if required. The values apply only for this segment. Touch the arrow to edit the data. The corresponding option window is opened. (>>> *10.4.3.7 "Option window “Motion parameters” (CP spline segment)" Page 407*)
5	Collision detection for this segment • **ColDetect** box is hidden: The setting at the block is valid for this segment. • **OFF**: The collision detection is switched off. • **CDSet_Set[***No.***]**: The collision detection is switched on. The values from data set *No.* are used for detection. (>>> *8.14.6 "Activating/deactivating collision detection via inline form" Page 348*)

Item	Description
6	Name of the data set containing logic parameters. The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the data. The corresponding option window is opened. (>>> *10.4.3.9 "Option window “Logic parameters”" Page 409*)

10.4.3.3 Programming an SCIRC segment

Precondition

- User rights: Function group **New motion range inline forms**
- A program is selected.
- T1 mode
- The CP spline block fold is open.

Procedure

1. Move the TCP to the auxiliary point.
2. Position the cursor in the line after which the segment is to be inserted in the spline block.
3. Select the menu sequence **Commands** > **Motion** > **SCIRC**.
4. Set the parameters in the inline form.
5. Press **Teach Aux**.
6. Move the TCP to the end point.
7. Press **Cmd OK**.

10.4.3.4 Inline form SCIRC spline segment

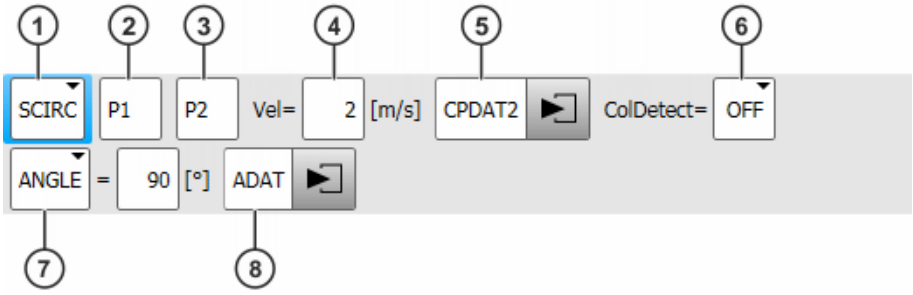


Fig. 10-14: Inline form SCIRC spline segment

By default, not all boxes of the inline form are displayed. The boxes can be displayed or hidden using the **Switch parameter** button.

Item	Description
1	Motion type **SCIRC**
2	Point name for the auxiliary point. The system automatically generates a name. The name can be overwritten.
3	Point name for the end point. The system automatically generates a name. The name can be overwritten.

Item	Description
4	Cartesian velocity By default, the value that is valid for the spline block is also valid for the segment. A separate value can be assigned here for the segment if required. The value applies only for this segment. • **0.001 … 2 m/s**
5	Name for the motion data set. The system automatically generates a name. The name can be overwritten. By default, the values that are valid for the spline block are also valid for the segment. Separate values can be assigned here for the segment if required. The values apply only for this segment. Touch the arrow to edit the data. The corresponding option window is opened. (>>> *10.4.3.7 "Option window "Motion parameters" (CP spline segment)" Page 407*)
6	Collision detection for this segment • **ColDetect** box is hidden: The setting at the block is valid for this segment. • **OFF**: The collision detection is switched off. • **CDSet_Set[***No.***]**: The collision detection is switched on. The values from data set *No.* are used for detection. (>>> *8.14.6 "Activating/deactivating collision detection via inline form" Page 348*)
7	Circular angle • **- 9,999° … + 9,999°** If a value less than -400° or greater than +400° is entered, a request for confirmation is generated when the inline form is saved asking whether entry is to be confirmed or rejected.
8	Name of the data set containing logic parameters. The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the data. The corresponding option window is opened. (>>> *10.4.3.9 "Option window "Logic parameters"" Page 409*)

10.4.3.5 Programming an SPTP segment

Precondition

- User rights: Function group **New motion range inline forms**
- A program is selected.
- T1 mode
- The PTP spline block fold is open.

Procedure

1. Move the TCP to the end point.
2. Position the cursor in the line after which the segment is to be inserted in the spline block.

3. Select the menu sequence **Commands** > **Motion** > **SPTP**.
4. Set the parameters in the inline form.
   (>>> *10.4.3.6 "Inline form for SPTP segment" Page 406*)
5. Press **Cmd OK**.

10.4.3.6 Inline form for SPTP segment

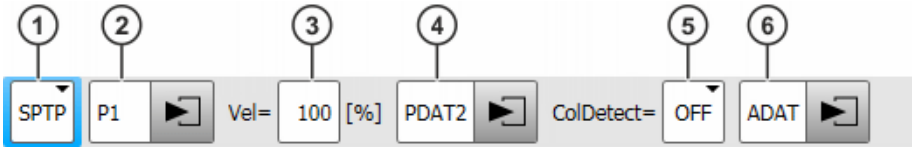


Fig. 10-15: Inline form for SPTP segment

By default, not all boxes of the inline form are displayed. The boxes can be displayed or hidden using the **Switch parameter** button.

Item	Description
1	Motion type **SPTP**
2	Point name for end point. The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the **Global point** setting. The corresponding window is opened.
3	Axis velocity By default, the value that is valid for the spline block is also valid for the segment. A separate value can be assigned here for the segment if required. The value applies only for this segment. • **1 … 100%**
4	Name for the motion data set. The system automatically generates a name. The name can be overwritten. By default, the values that are valid for the spline block are also valid for the segment. Separate values can be assigned here for the segment if required. The values apply only for this segment. Touch the arrow to edit the point data. The corresponding option window is opened. (>>> *10.4.3.8 "Option window “Motion parameters” (SPTP)" Page 408*)
5	Collision detection for this segment • **ColDetect** box is hidden: The setting at the block is valid for this segment. • **OFF**: The collision detection is switched off. • **CDSet_Set[***No.***]**: The collision detection is switched on. The values from data set *No.* are used for detection. (>>> *8.14.6 "Activating/deactivating collision detection via inline form" Page 348*)

Item	Description
6	Name of the data set containing logic parameters. The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the data. The corresponding option window is opened. (>>> *10.4.3.9 "Option window "Logic parameters"" Page 409*)

10.4.3.7 Option window "Motion parameters" (CP spline segment)

Motion parameters

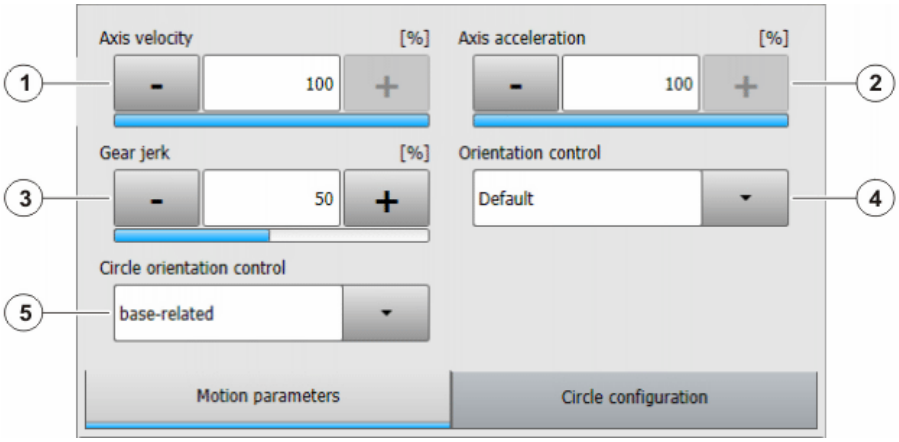


Fig. 10-16: Option window "Motion parameters" (CP spline segment)

Item	Description
1	Axis velocity. The value refers to the maximum value specified in the machine data. • **1 … 100%**
2	Axis acceleration. The value refers to the maximum value specified in the machine data. • **1 … 100%**
3	Gear jerk. The jerk is the change in acceleration. The value refers to the maximum value specified in the machine data. • **1 … 100%**
4	Orientation control selection.
5	Only in the case of SCIRC segments: Orientation control reference system selection.

Circle configuration

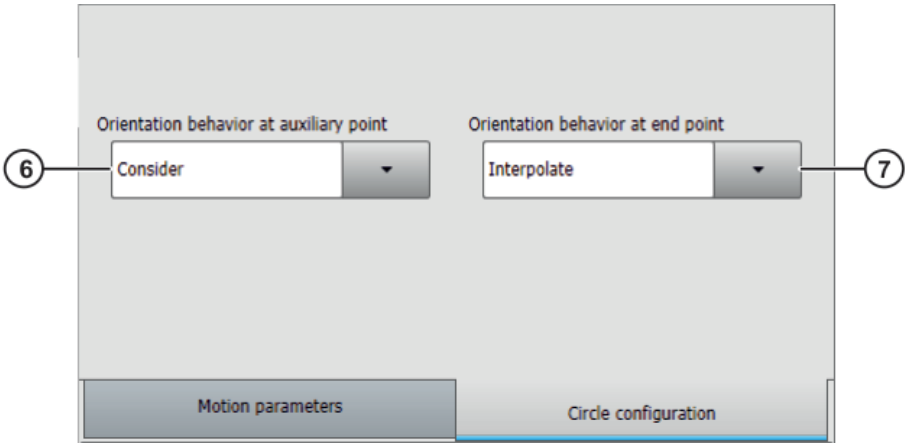


Fig. 10-17: Circle configuration (SCIRC segment)

Item	Description
6	Only in the case of SCIRC segments: Selection of orientation behavior at auxiliary point.
7	Only in the case of SCIRC segments: This box is only displayed if **ANGLE** was selected in the inline form. Selection of orientation behavior at end point.

10.4.3.8 Option window “Motion parameters” (SPTP)

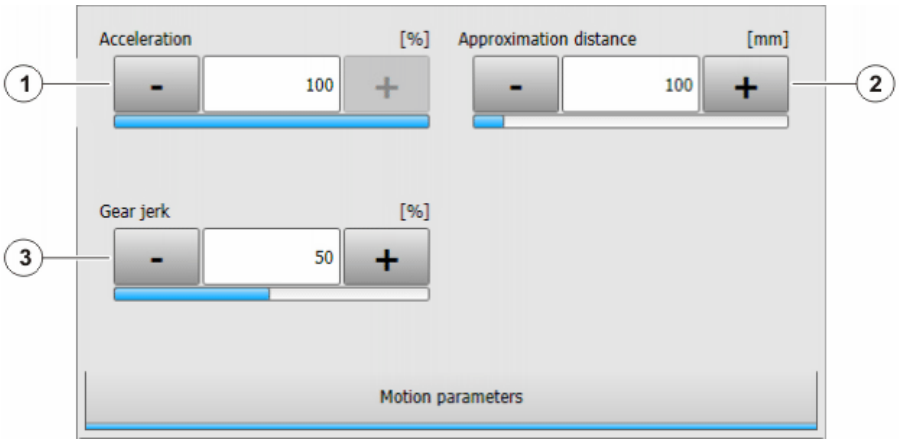


Fig. 10-18: Option window “Motion parameters” (SPTP)

Item	Description
1	Axis acceleration. The value refers to the maximum value specified in the machine data. • **1 … 100%**
2	This box is not available for SPTP segments. In the case of individual SPTP motions, this box is only displayed if **CONT** was selected in the inline form. Furthest distance before the end point at which approximate positioning can begin. The maximum permissible value is half the distance between the start point and the end point. If a higher value is entered, this is ignored and the maximum value is used.

Item	Description
3	Gear jerk. The jerk is the change in acceleration. The value refers to the maximum value specified in the machine data. • **1 … 100%**

10.4.3.9 Option window “Logic parameters”

Trigger

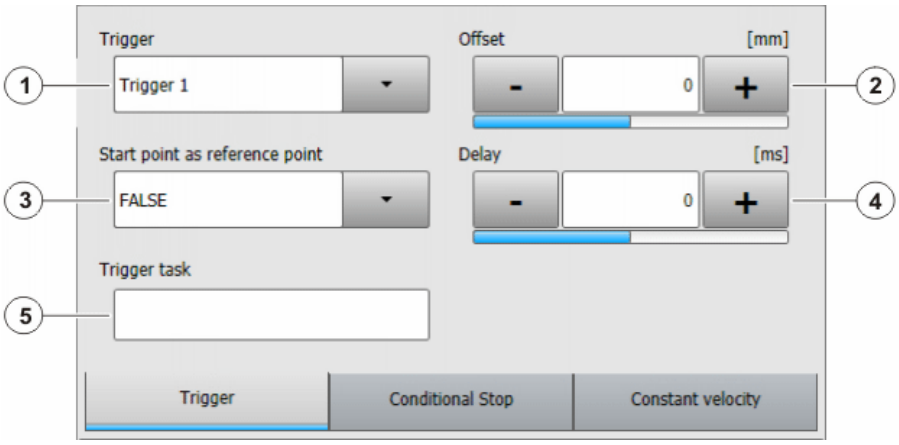


Fig. 10-19: Trigger

Item	Description
1	A (further) trigger can be assigned to the motion by means of the button **Select action** > **Add trigger**. If it is the first trigger for this motion, this command also causes the **Trigger** box to be displayed. A maximum of 8 triggers per motion are possible. (A trigger can be removed again by means of **Select action** > **Remove trigger**.)
2	Reference point of the trigger • **TRUE**: Start point • **FALSE**: End point (>>> *11.11.2.1 "Reference point for approximate positioning – overview" Page 537*)
3	Spatial offset relative to the end or start point • Negative value: Offset towards the start of the motion • Positive value: Offset towards the end of the motion (>>> *"Max. offset" Page 534*) The shift in space can also be taught. In this case, the box **Start point is reference point** is automatically set to **FALSE**. (>>> *10.4.3.10 "Teaching the shift in space for logic parameters" Page 412*)

Item	Description
4	Shift in time relative to **Offset** • Negative value: Shift towards the start of the motion. • Positive value: Trigger is switched after *Time* has elapsed. (>>> *"Max. offset" Page 534*)
5	Statement that is to be initiated by the trigger. The following are possible: • Assignment of a value to a variable **Note:** There must be no runtime variable on the left-hand side of the assignment. • OUT statement; PULSE statement; CYCFLAG statement • Subprogram call. In this case, the priority must be specified. Example: `my_subprogram() PRIO = 81` Priorities 1, 2, 4 to 39 and 81 to 128 are available. Priorities 40 to 80 are reserved for cases in which the priority is automatically assigned by the system. If the priority is to be assigned automatically by the system, the following is programmed: `PRIO = -1`. If several triggers call subprograms at the same time, the trigger with the highest priority is processed first, then the triggers of lower priority. 1 = highest priority.

Conditional stop

Further information about the conditional stop can be found in this documentation.
(>>> *10.4.5 "Conditional stop" Page 419*)

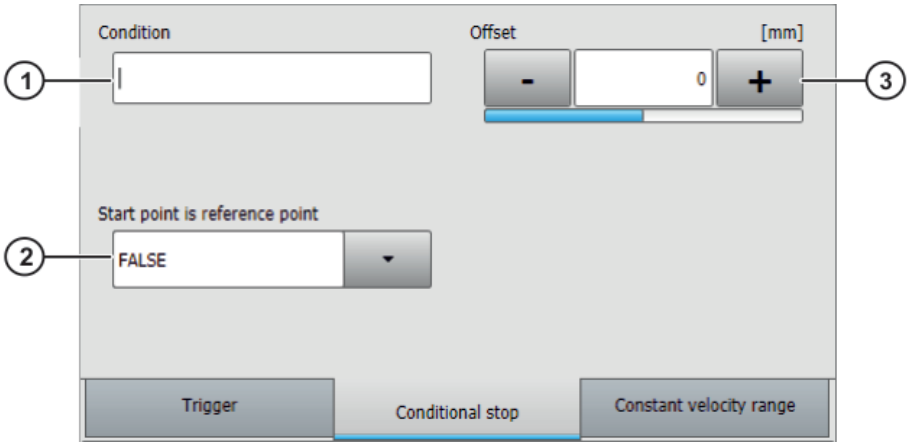


Fig. 10-20: Conditional stop

Item	Description
1	Stop condition. The following are permitted: • a global Boolean variable • a signal name • a comparison • a simple logic operation: NOT, OR, AND or EXOR
2	The conditional stop can refer to either the start point or the end point of the motion. • **TRUE**: Start point • **FALSE**: End point If the reference point is approximated, the same rules apply as for the PATH trigger. (>>> *11.11.2.1 "Reference point for approximate positioning – overview" Page 537*)
3	The stop point can be shifted in space. For this, the desired distance from the start or end point must be specified. If no shift in space is desired, enter “0”. • Positive value: Offset towards the end of the motion • Negative value: Offset towards the start of the motion There are limits to the distance the stop point can be offset. The same limits apply as for the PATH trigger. (>>> *"Max. offset" Page 534*) The shift in space can also be taught. In this case, the box **Start point is reference point** is automatically set to **FALSE**. (>>> *10.4.3.10 "Teaching the shift in space for logic parameters" Page 412*)

Constant velocity range

Constant velocity range is only available for CP spline segments.

Basic information about the constant velocity range can be found here: (>>> *10.4.6 "Constant velocity range in the CP spline block" Page 423*)

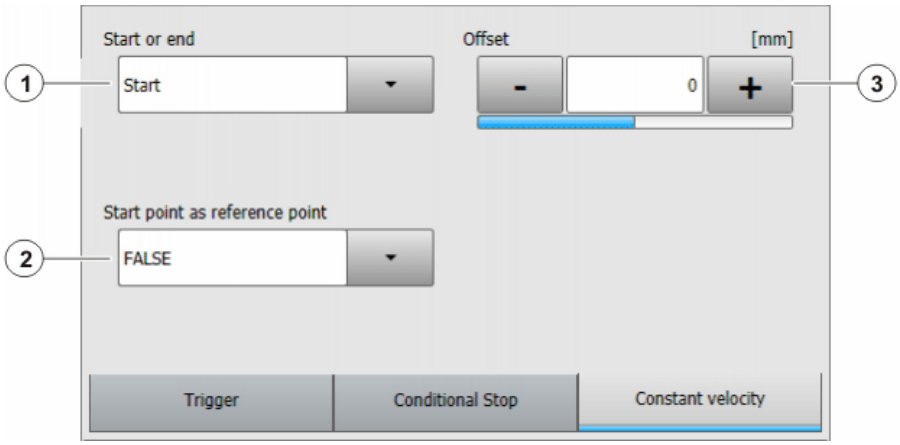


Fig. 10-21: Constant velocity range

Item	Description
1	• **Start**: Defines the start of the constant velocity range. • **End**: Defines the end of the constant velocity range.
2	**Start** and **End** can refer to either the start point or the end point of the motion. • **TRUE**: **Start** or **End** refers to the start point. If the start point is approximated, the reference point is generated in the same way as for homogenous approximate positioning with the PATH trigger. (>>> *11.11.2.2 "Reference point for homogenous approximate positioning" Page 537*) • **FALSE**: **Start** or **End** refers to the end point. If the end point is approximated, **Start** or **End** refers to the start of the approximate positioning arc.
3	The start or end of the constant velocity range can be shifted in space. For this, the desired distance must be specified. If no shift in space is desired, enter “0”. • Positive value: Offset towards the end of the motion • Negative value: Offset towards the start of the motion (>>> *10.4.6.2 "Maximum limits" Page 423*) The shift in space can also be taught. In this case, the **Start point is reference point** box is automatically set to **FALSE**. (>>> *10.4.3.10 "Teaching the shift in space for logic parameters" Page 412*)

10.4.3.10 Teaching the shift in space for logic parameters

Description

Shifts in space can be specified in the option window **Logic parameters** for trigger, conditional stop and constant velocity range. Instead of entering these offsets numerically, they can also be taught.

> If an offset is taught, the box **Start point is reference point** in the corresponding tab is automatically set to **FALSE**, as the taught distance refers to the end point of the motion.

Precondition

- A program is selected.
- Operating mode T1
- The point for which the offset is to apply has already been taught.

Procedure

1. Move the TCP to the desired position.
2. Position the cursor in the line containing the motion instruction for which the offset is to be taught.
3. Press **Change**. The inline form for this instruction is opened.
4. Open the option window **Logic parameters** and select the required tab.
5. Press **Select action** then one of the following buttons depending on what the offset is to be taught for:

- **Teach trigger path**
- **Teach conditional stop path**
- **Teach constant velocity range path**

The distance from the end point of the current motion statement is now applied as the value for the offset.

6. Save the change by pressing **Cmd OK**.

10.4.4 Programming individual spline motions

10.4.4.1 Programming an individual SLIN motion

Precondition

- User rights: Function group **New motion range inline forms**
- A program is selected.
- T1 mode

Procedure

1. Move the TCP to the end point.
2. Position the cursor in the line after which the motion is to be inserted.
3. Select **Commands** > **Motion** > **SLIN**.
4. Set the parameters in the inline form.
   (>>> *10.4.4.2 "Inline form “SLIN”" Page 413*)
5. Press **Cmd OK**.

10.4.4.2 Inline form “SLIN”

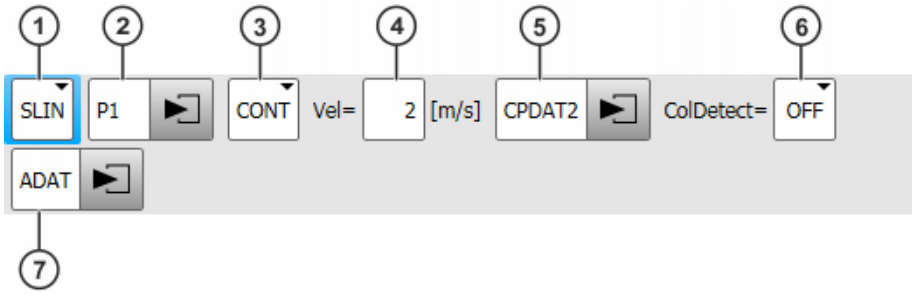


Fig. 10-22: Inline form “SLIN” (individual motion)

Item	Description
1	Motion type **SLIN**
2	Point name for end point. The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the point data. The corresponding option window is opened. (>>> *10.3.7 "Option window “Frames”" Page 395*) The arrow can also be used to edit the **Global point** setting.
3	• **CONT**: end point is approximated. • **[Empty box]**: The motion stops exactly at the end point.

Item	Description
4	Velocity • **0.001 … 2 m/s**
5	Name for the motion data set. The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the point data. The corresponding option window is opened. (>>> *10.4.4.3 "Option window “Motion parameters” (SLIN)" Page 414*)
6	Collision detection for this motion • **OFF**: The collision detection is switched off. • **CDSet_Set[***No.***]**: The collision detection is switched on. The values from data set *No.* are used for detection. (>>> *8.14.6 "Activating/deactivating collision detection via inline form" Page 348*)
7	This box can be displayed or hidden by means of **Spline logic**. Name of the data set containing logic parameters. The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the data. The corresponding option window is opened. (>>> *10.4.3.9 "Option window “Logic parameters”" Page 409*)

10.4.4.3 Option window “Motion parameters” (SLIN)

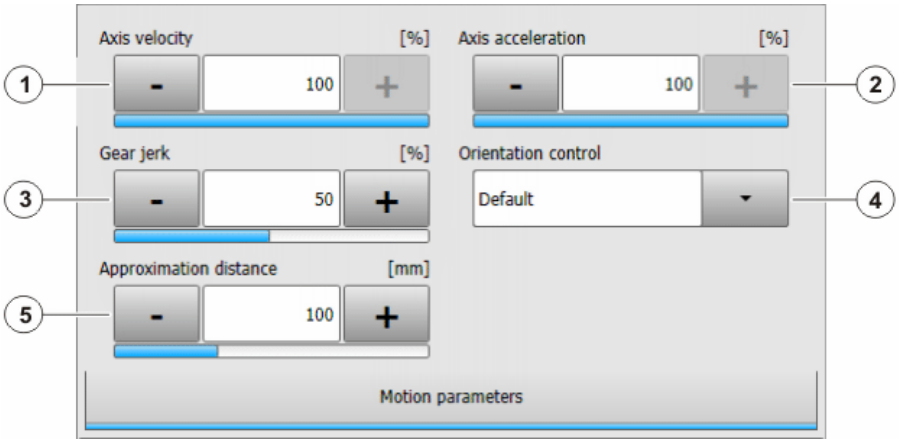


Fig. 10-23: Option window “Motion parameters” (SLIN)

Item	Description
1	Axis velocity. The value refers to the maximum value specified in the machine data. • **1 … 100%**
2	Axis acceleration. The value refers to the maximum value specified in the machine data. • **1 … 100%**

Item	Description
3	Gear jerk. The jerk is the change in acceleration. The value refers to the maximum value specified in the machine data. • **1 … 100%**
4	Orientation control selection.
5	This box is only displayed if **CONT** was selected in the inline form. Furthest distance before the end point at which approximate positioning can begin. The maximum permissible value is half the distance between the start point and the end point. If a higher value is entered, this is ignored and the maximum value is used.

10.4.4.4 Programming an individual SCIRC motion

Precondition

- User rights: Function group **New motion range inline forms**
- A program is selected.
- T1 mode

Procedure

1. Move the TCP to the auxiliary point.
2. Position the cursor in the line after which the motion is to be inserted.
3. Select the menu sequence **Commands** > **Motion** > **SCIRC**.
4. Set the parameters in the inline form.
   (>>> *10.4.4.5 "Inline form “SCIRC”" Page 415*)
5. Press **Teach Aux**.
6. Move the TCP to the end point.
7. Press **Cmd OK**.

10.4.4.5 Inline form “SCIRC”

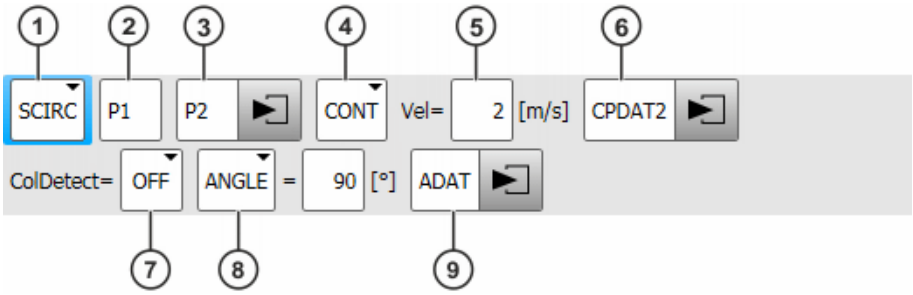


Fig. 10-24: Inline form “SCIRC” (individual motion)

Item	Description
1	Motion type **SCIRC**
2	Point name for the auxiliary point. The system automatically generates a name. The name can be overwritten.

Item	Description
3	Point name for the end point. The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the point data. The corresponding option window is opened. (>>> *10.3.7 "Option window "Frames"" Page 395*)
4	• **CONT**: end point is approximated. • **[Empty box]**: The motion stops exactly at the end point.
5	Velocity • **0.001 … 2 m/s**
6	Name for the motion data set. The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the point data. The corresponding option window is opened. (>>> *10.4.4.6 "Option window "Motion parameters" (SCIRC)" Page 417*)
7	Collision detection for this motion • **OFF**: The collision detection is switched off. • **CDSet_Set[***No.***]**: The collision detection is switched on. The values from data set *No.* are used for detection. (>>> *8.14.6 "Activating/deactivating collision detection via inline form" Page 348*)
8	Circular angle • **- 9,999° … + 9,999°** If a circular angle less than -400° or greater than +400° is entered, a request for confirmation is generated when the inline form is saved asking whether entry is to be confirmed or rejected.
9	This box can be displayed or hidden by means of **Spline logic**. Name of the data set containing logic parameters. The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the data. The corresponding option window is opened. (>>> *10.4.3.9 "Option window "Logic parameters"" Page 409*)

10.4.4.6 Option window “Motion parameters” (SCIRC)

Motion parameters

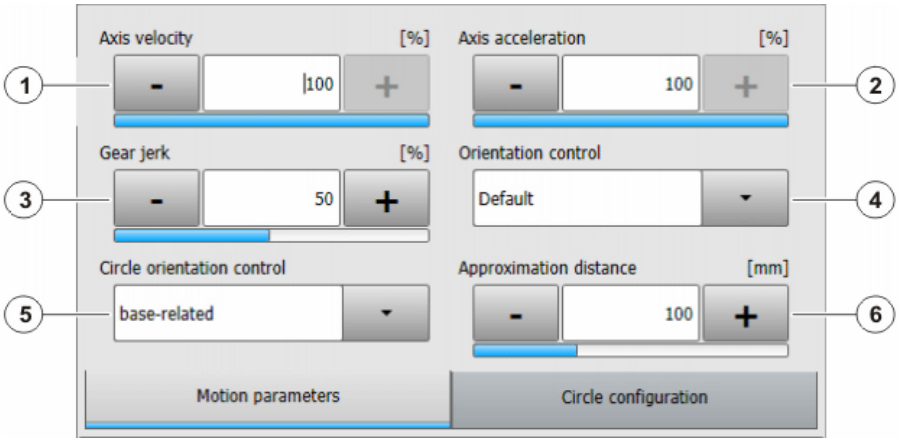


Fig. 10-25: Motion parameters (SCIRC)

Item	Description
1	Axis velocity. The value refers to the maximum value specified in the machine data. • **1 … 100%**
2	Axis acceleration. The value refers to the maximum value specified in the machine data. • **1 … 100%**
3	Gear jerk. The jerk is the change in acceleration. The value refers to the maximum value specified in the machine data. • **1 … 100%**
4	Orientation control selection.
5	Orientation control reference system selection.
6	This box is only displayed if **CONT** was selected in the inline form. Furthest distance before the end point at which approximate positioning can begin. The maximum permissible value is half the distance between the start point and the end point. If a higher value is entered, this is ignored and the maximum value is used.

Circle configuration

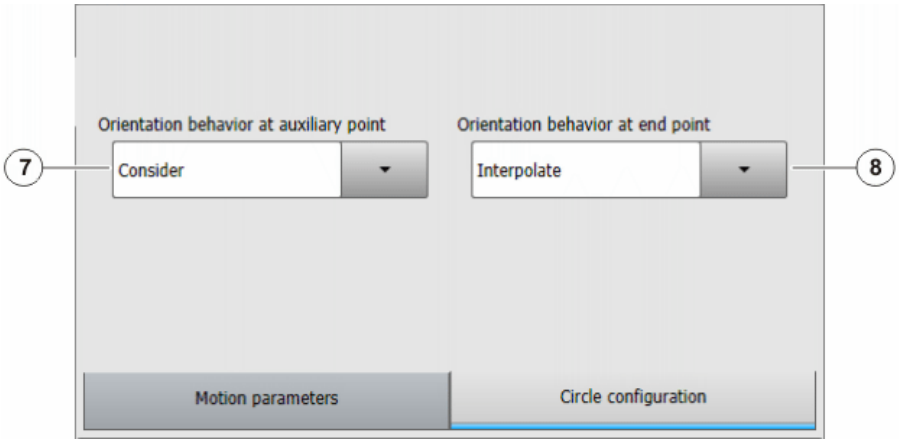


Fig. 10-26: Circle configuration (SCIRC)

Item	Description
7	Selection of orientation behavior at auxiliary point.
8	This box is only displayed if **ANGLE** was selected in the inline form. Selection of orientation behavior at end point.

10.4.4.7 Programming an individual SPTP motion

Precondition

- User rights: Function group **New motion range inline forms**
- A program is selected.
- T1 mode

Procedure

1. Move the TCP to the end point.
2. Position the cursor in the line after which the motion is to be inserted.
3. Select **Commands** > **Motion** > **SPTP**.
4. Set the parameters in the inline form.
   (>>> *10.4.4.8 "Inline form “SPTP”" Page 418*)
5. Press **Cmd OK**.

10.4.4.8 Inline form “SPTP”

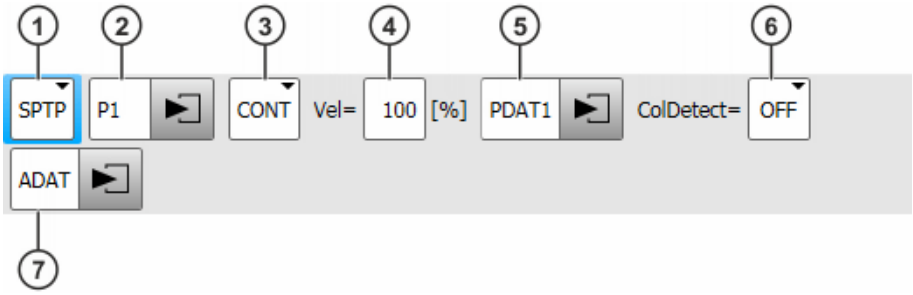


Fig. 10-27: Inline form “SPTP” (individual motion)

Item	Description
1	Motion type **SPTP**
2	Point name for end point. The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the point data. The corresponding option window is opened. (>>> *10.3.7 "Option window “Frames”" Page 395*) The arrow can also be used to edit the **Global point** setting.
3	• **CONT**: end point is approximated. • **[Empty box]**: The motion stops exactly at the end point.
4	Velocity • **1 … 100%**
5	Name for the motion data set. The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the point data. The corresponding option window is opened. (>>> *10.4.3.8 "Option window “Motion parameters” (SPTP)" Page 408*)
6	Collision detection for this motion • **OFF**: The collision detection is switched off. • **CDSet_Set[***No.***]**: The collision detection is switched on. The values from data set *No.* are used for detection. (>>> *8.14.6 "Activating/deactivating collision detection via inline form" Page 348*)
7	This box can be displayed or hidden by means of **Spline logic**. Name of the data set containing logic parameters. The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the data. The corresponding option window is opened. (>>> *10.4.3.9 "Option window “Logic parameters”" Page 409*)

10.4.5 Conditional stop

Description

The conditional stop enables the user to define a point on the path at which the robot stops if a certain condition is met. This point is called the “stop point”. As soon as the condition is no longer met, the robot resumes its motion.

During the runtime, the robot controller calculates the latest point at which the robot must brake in order to be able to stop at the stop point. From this point (braking point) onwards, it monitors whether or not the condition is met.

- If the condition is met at the braking point, the robot brakes in order to stop at the stop point.

  If, however, the condition then switches back to “not met” before the stop point is reached, the robot accelerates again and does not stop.
- If the condition is not met at the braking point, the robot motion is continued without braking.

Essentially, any number of conditional stops can be programmed. A maximum of 10 “braking point → stop point” paths may overlap, however.

While the robot is braking, the robot controller displays the following message in T1/T2 mode: *Conditional stop active (line {Line number}).*

(>>> *10.4.5.2 "Stop condition: example and braking characteristics" Page 421*)

Programming

Programming with KRL syntax:

- using the statement STOP WHEN PATH

Programming with inline forms:

- In the spline block (CP and PTP) or in the individual spline block: in the option window “**Logic parameters**”
- Before a spline block (CP and PTP): via the inline form **Spline Stop Condition**

10.4.5.1 Inline form “Spline Stop Condition”

This inline form may only be used before a spline block. There may be other statements between the inline form and the spline block, but no motion instructions.

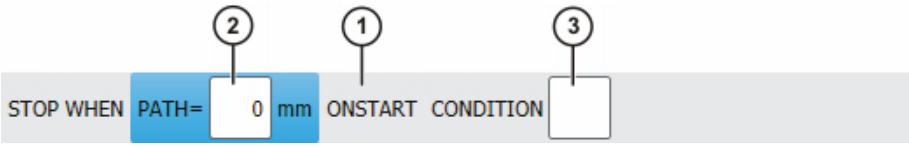


Fig. 10-28: Inline form “Spline Stop Condition”

Item	Description
1	Point to which the conditional stop refers • With ONSTART: last point before the spline block • Without ONSTART: last point in the spline block If the spline is approximated, the same rules apply as for the PATH trigger. **Note:** Information about approximate positioning with the PATH trigger is contained in the Operating and Programming Instructions for System Integrators. ONSTART can be set or removed using the **Toggle OnStart** button.
2	The stop point can be shifted in space. For this, the desired distance from the reference point must be specified. If no shift in space is desired, enter “0”. • Positive value: Offset towards the end of the motion • Negative value: Offset towards the start of the motion There are limits to the distance the stop point can be offset. The same limits apply as for the PATH trigger. The shift in space can also be taught. (>>> *"Teach path" Page 421*)

Item	Description
3	Stop condition The following are permitted: • a global Boolean variable • a signal name • a comparison • a simple logic operation: NOT, OR, AND or EXOR

Teach path

Button	Description
Teach path	If an offset is desired, it is not necessary to enter the value into the inline form numerically; the offset can also be taught. This is done via **Teach path**. If an offset is taught, ONSTART is automatically removed, if set in the inline form, as the taught distance always refers to the end point of the motion. The teaching sequence is the same as that for the option window **Logic parameters**. (>>> *10.4.3.10 "Teaching the shift in space for logic parameters" Page 412*)

10.4.5.2 Stop condition: example and braking characteristics

Example

The indentations are not present by default and have been inserted here for greater clarity.

```
PTP P0 Vel=100 % PDAT1 Tool[1] Base[0]

SPLINE S1 Vel=2 m/s CPDAT1 Tool[1] Base[0]

SPL P1 ADAT1

  STOP WHEN PATH = 50 IF $in[77]==FALSE

  SPL XP1

SPL P2

SPL P3

ENDSPLINE
```

Fig. 10-29: Inline programming example (folds expanded)

Line	Description
4	If the input $IN[77] is FALSE, the robot stops 50 mm after P2 and waits until $IN[77] is TRUE.

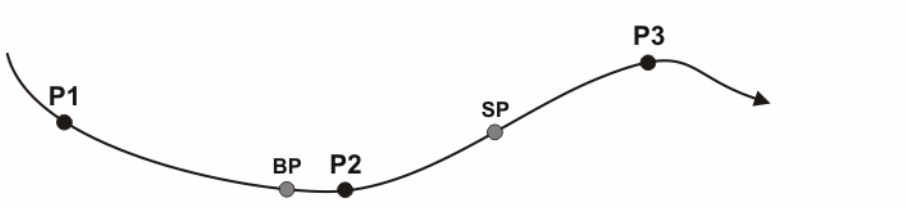


Fig. 10-30: Example of STOP WHEN PATH

Point	Description
BP	**B**raking **P**oint: The robot must start braking here in order to stop at the stop point. From this point onwards, the robot controller monitors whether or not the stop condition is met. The position of **BP** depends on the velocity and the override setting and cannot be identified by the operator.
SP	**S**top **P**oint The distance **P2** → **SP** is 50 mm long.

Braking characteristics

Situation at BP	Behavior of the robot
$IN[77] == FALSE	The robot brakes and stops at **SP**.
$IN[77] == TRUE	The robot does not brake and does not stop at **SP**. The program is executed as if the STOP WHEN PATH statement were not present.
1. $IN[77] == FALSE at **BP**. 2. The input switches to TRUE between **BP** and **SP**.	1. The robot brakes at **BP**. 2. If the input is TRUE, the robot accelerates again and does not stop at **SP**.
1. $IN[77] == TRUE at **BP**. 2. The input switches to FALSE between **BP** and **SP**.	1. The robot does not brake at **BP**. 2. If the input is FALSE, the robot stops with a path-maintaining EMERGENCY STOP and comes to a standstill at an unpredictable point.

If the stop condition is not met until the robot has already passed **BP**, it is too late to stop at **SP** with a normal braking ramp. In this case, the robot stops with a path-maintaining EMERGENCY STOP and comes to a standstill at an unpredictable point.

- If the EMERGENCY STOP causes the robot to stop after **SP**, the program cannot be resumed until the stop condition is no longer met.
- If the path-maintaining EMERGENCY STOP causes the robot to stop before **SP**, the following occurs when the program is resumed:
  - If the stop condition is no longer met, the robot resumes its motion.
  - If the stop condition is still met, the robot moves as far as **SP** and stops there.

10.4.6 Constant velocity range in the CP spline block

Description

In a CP spline block, a range can be defined in which the robot maintains the programmed velocity constant where possible. This range is called the “constant velocity range”.

- 1 constant velocity range can be defined per CP spline block.
- A constant velocity range is defined by a start statement and an end statement.
- The range cannot extend beyond the spline block.
- There is no lower limit to the size of the range.

If it is not possible to maintain the programmed velocity constant, the robot controller indicates this by means of a message during program execution.

Constant velocity range over several segments:

A constant velocity range can extend over several segments with different programmed velocities. In this case, the lowest of the velocities is valid for the whole range.

Even in the segments with a higher programmed velocity, the motion is executed with the lowest velocity in this case. No message is generated indicating that the velocity has not been maintained. This only occurs if the lowest velocity cannot be maintained.

Programming

A constant velocity range can be programmed in the following ways:

- If programming with KRL syntax: by means of the statement CONST_VEL
- If programming with inline forms:

  The start or end of the range is stored in the corresponding CP segment in the option window **Logic parameters**.

10.4.6.1 Block selection to the constant velocity range

Description

If a block selection to a constant velocity range is carried out, the robot controller ignores it and generates a corresponding message. The motions are executed as if no constant velocity range were programmed.

A block selection to the path section defined by the offset values is considered as a block selection to the constant velocity range. The motion blocks in which the start and end of the range are programmed, however, are irrelevant.

10.4.6.2 Maximum limits

If the start or end point of the spline block is an exact positioning point:

- The constant velocity range starts at the start point at the earliest.
- The constant velocity range ends at the end point at the latest.

If the offset value is such that these limits would be exceeded, the robot controller automatically reduces the offset and generates the following message: *CONST_VEL {Start/End} = {Offset} cannot be implemented; {New offset} will be used.*

The robot controller reduces the offset far enough to create a range in which the constant programmed velocity can be maintained. In other words, it does not necessarily shift the limit exactly to the start or end point of the spline block, but possibly further inwards.

The same message is generated if the range is already in the spline block beforehand, but the defined velocity cannot be maintained due to the offset. In this case, once again, the robot controller reduces the offset.

If the start or end point of the spline block is approximated:

- The constant velocity range starts at the start of the approximate positioning arc of the start point at the earliest.
- The constant velocity range ends at the start of the approximate positioning arc of the end point at the latest.

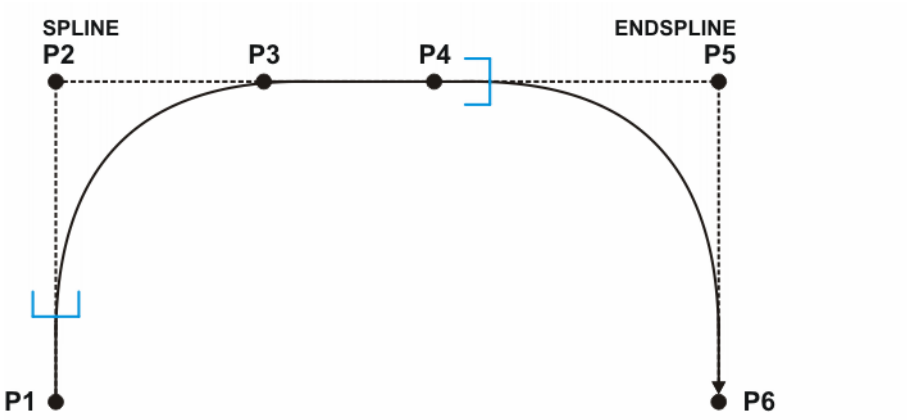


Fig. 10-31: Maximum limits for approximated SPLINE/ENDSPLINE

If the offset is such that these limits would be exceeded, the robot controller automatically sets the limit to the start of the corresponding approximate positioning arc. It does not generate a message.

10.5 Displaying the distance between points

Precondition

- User rights: Function group **General KRL program changes**
- T1 or T2 mode
- Program is selected or open.

Procedure

1. Select the points (= the motion blocks) for which the distance is to be displayed. Multiple consecutive blocks can also be selected.
2. Select the menu sequence **Edit** > **Marked region** > **Cart. distance**.

   A window opens. It displays the following information:

   - The Cartesian coordinates of the first selected point
   - The Cartesian coordinates of the last selected point
   - The distance between the coordinates in millimeters and degrees
   - The distance between the TCP position at the first and last point in millimeters and degrees
3. Select other points if required.
4. Press **Refresh** to update the display.

10.6 Modifying programmed motions

10.6.1 Modifying motion parameters

Precondition

- A program is selected.
- Operating mode T1

Procedure

1. Position the cursor in the line containing the instruction that is to be changed.
2. Press **Change**. The inline form for this instruction is opened.
3. Modify parameters.
4. Save changes by pressing **Cmd Ok**.

10.6.2 Modifying blocks of motion parameters

Precondition

- User rights: Function group **General KRL program changes**
- T1 mode
- A program is selected.

Procedure

1. Select the motion instructions to be modified. (Only consecutive motion instructions can be modified as a block.)
2. Press **Change**. The inline form of the first selected motion block opens.
3. Modify parameters.
4. Press **Cmd OK**. The changes will be applied to the selected motion blocks where possible.

   Some changes will not be applied in every motion block, e.g. it is not possible to apply the PTP parameter **Velocity** in a LIN motion block.

10.6.3 Re-teaching a point

Description

The coordinates of a taught point can be modified. This is done by moving to the new position and overwriting the old point with the new position.

Precondition

- A program is selected.
- Operating mode T1

Procedure

1. Move the TCP to the desired position.
2. Position the cursor in the line containing the motion instruction that is to be changed.
3. Press **Change**. The inline form for this instruction is opened.

4. For PTP and LIN motions: Press **Touch Up** to accept the current position of the TCP as the new end point.

   For CIRC motions:

   - Press **Teach Aux** to accept the current position of the TCP as the new auxiliary point.
   - Or press **Teach End** to accept the current position of the TCP as the new end point.
5. Confirm the request for confirmation with **Yes**.
6. Save change by pressing **Cmd Ok**.

10.6.4 Transforming blocks of coordinates

Precondition

- User rights: Function group **General KRL program changes**
- T1 mode
- A program is selected.

Procedure

1. Select the motion instructions to be modified. (Only consecutive motion instructions can be modified as a block.)
2. Select the menu sequence **Edit** > **Marked region**. Select transformation type.

   The corresponding window is opened.

   (>>> *10.6.4.1 "“Axis mirroring” window" Page 429*)

   (>>> *10.6.4.2 "“Transform - Axis Specific” window" Page 430*)

   (>>> *10.6.4.3 "“Transform - Cartesian Base” window" Page 431*)
3. Enter values for the transformation and press **Calculate**.

Overview

The following transformation types are available:

- **Transform - Cartesian base**
- **Transform - Cartesian tool**
- **Transform - Cartesian World**
- **Transform - Axis-specific**
- **Axis mirroring**

Transform - Base

Transform - Cartesian Base:

The transformation refers to the current BASE coordinate system.

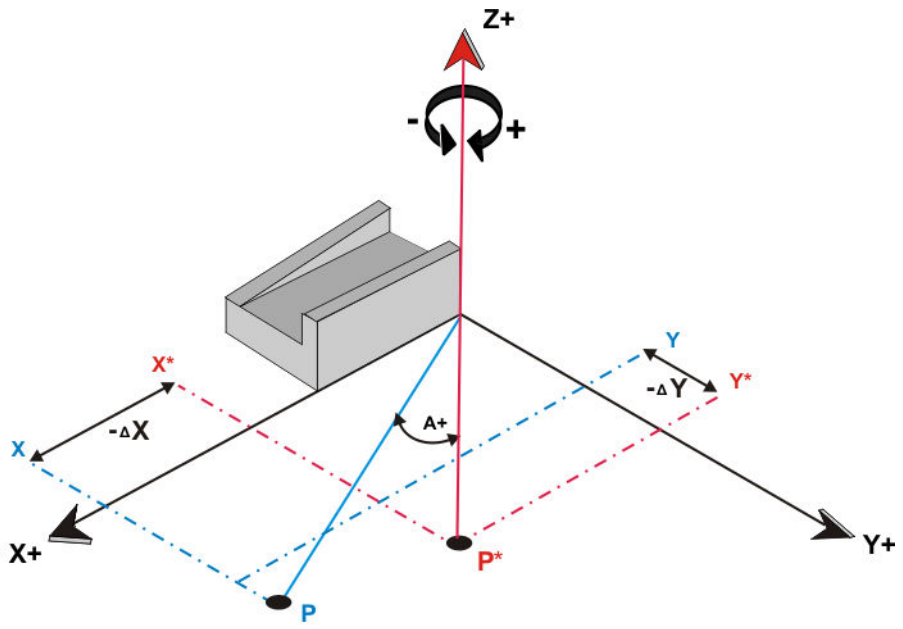


Fig. 10-32: Transform - Cartesian Base

Point P is offset by Δ X and Δ Y in the negative direction. The new position of the point is P*.

Transform - TCP

Transform - Cartesian TCP:

The transformation refers to the current TOOL coordinate system.

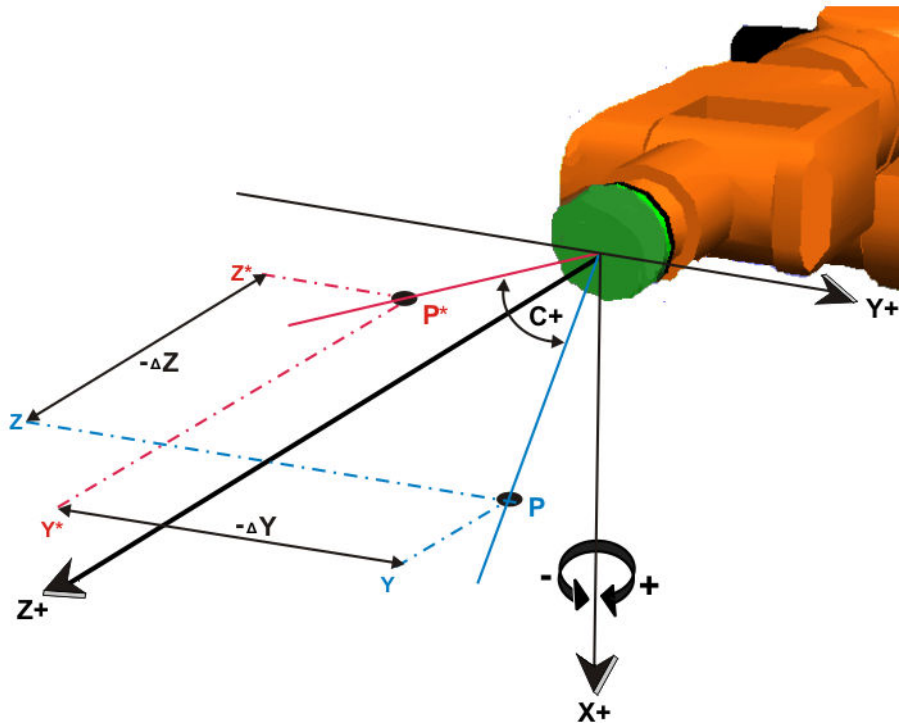


Fig. 10-33: Transform - Cartesian TCP

Point P is offset by Δ Z and Δ Y in the negative direction. The new position of the point is P*.

Transform - World

Transform - Cartesian World:
The transformation is relative to the WORLD coordinate system.

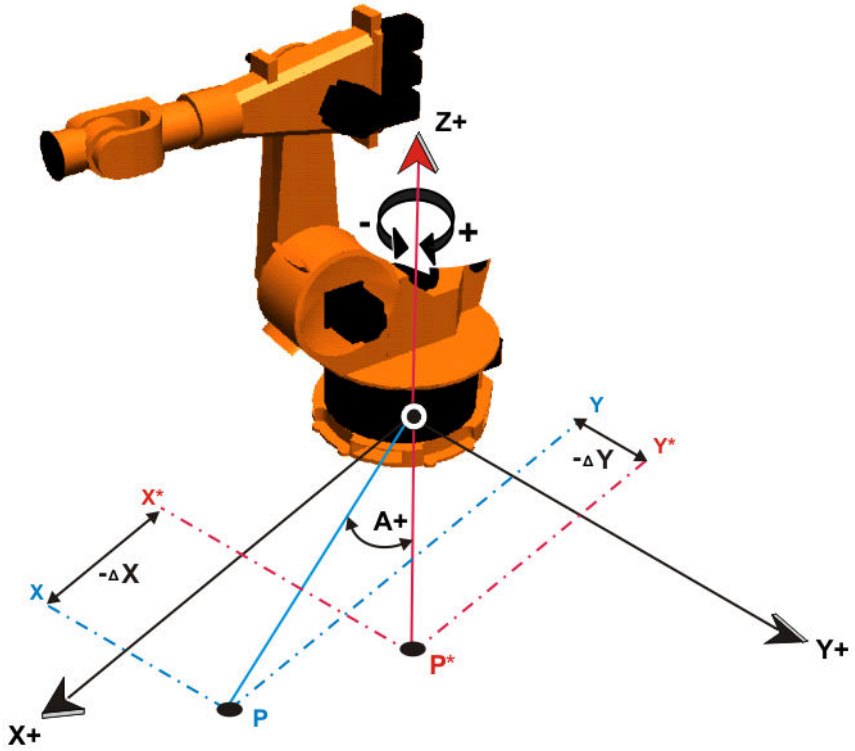


Fig. 10-34: Transform - Cartesian World

Point P is offset by Δ X and Δ Y in the negative direction. The new position of the point is P*.

Transform - Axis Specific

Transform - Axis Specific:
The transformation is axis-specific.

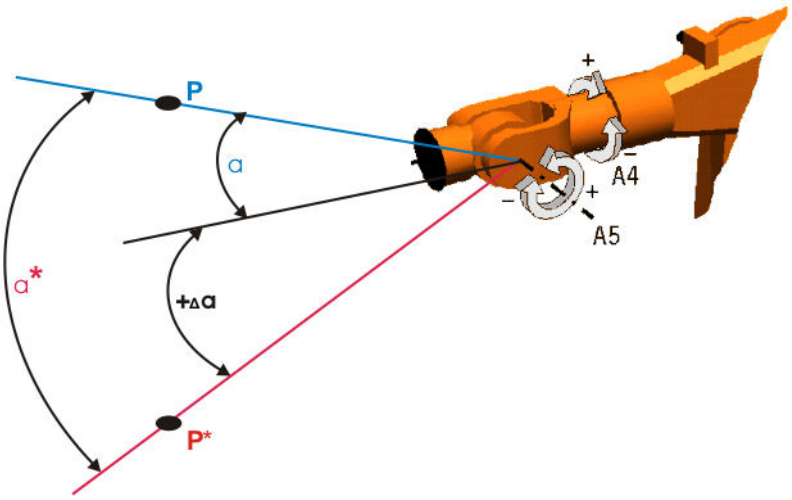


Fig. 10-35: Transform - Axis Specific

Axis A5 is rotated by the angle Δα. The new position of point P is P*.

Mirroring

Mirroring:

Mirroring in the XY plane of the ROBROOT coordinate system.

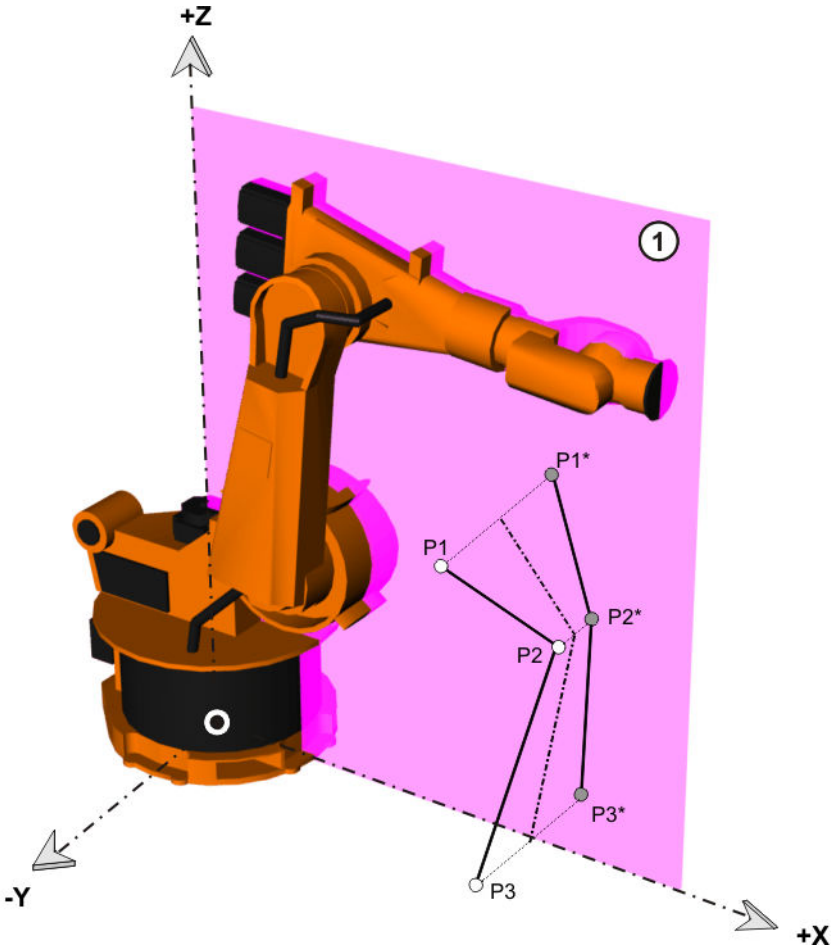


Fig. 10-36: Mirroring

Points P1, P2 and P3 are mirrored in the XY plane (1). The new positions of the points are P1*, P2* and P3*.

10.6.4.1 “Axis mirroring” window

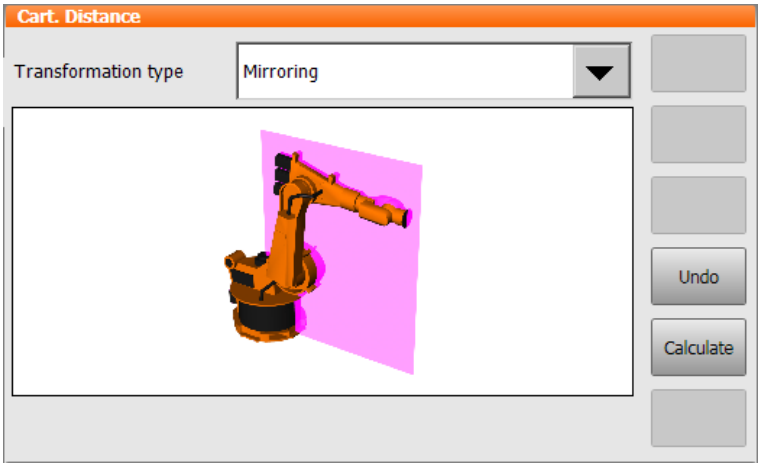


Fig. 10-37: Mirroring

No values need to be entered in this window. Pressing the **Calculate** button mirrors the point coordinates in the XZ plane of the ROBROOT coordinate system.

Following mirroring of the coordinates, the tool used must also be mirrored in the XZ plane.

The following buttons are available:

Button	Description
Calculate	Mirrors the coordinates of the selected motion points in the XZ plane, converts the coordinates to axis angles and applies the new values.
Undo	Undoes the mirroring and restores the old point data.

Only selected points with a complete E6POS definition are copied. This includes, for example, all those that were generated via inline forms during programming. Points without a complete E6POS definition are ignored during the point offset.

10.6.4.2 “Transform - Axis Specific” window

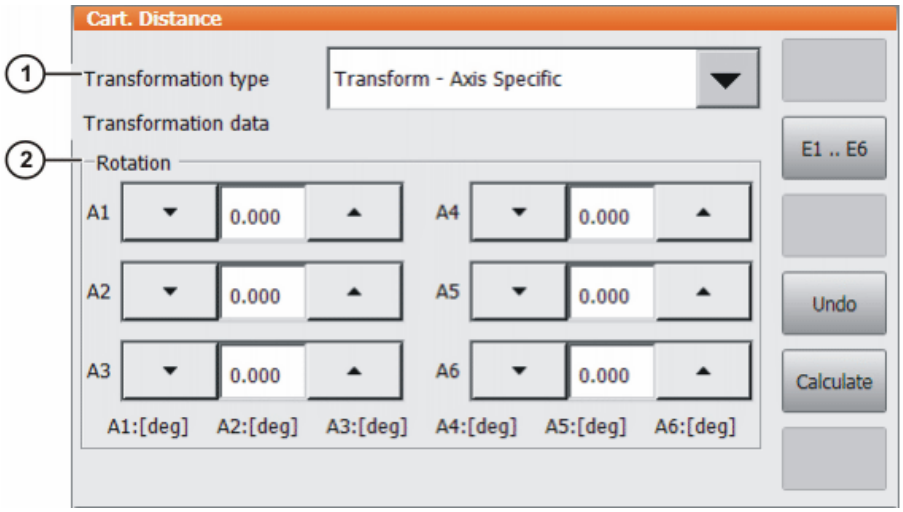


Fig. 10-38: Point transformation - axis-specific

Item	Description
1	Selection of the transformation type
2	**Rotation** group: input boxes for the position offset of axes A1 ... A6 • Range of values: Dependent on the configuration of the axis-specific workspaces **E1 .. E6** switches to the **External axes** group: input boxes for the position offset of axes E1 ... E6 **Note:** Values can only be entered for configured axes.

The following buttons are available:

Button	Description
E1 .. E6/A1 .. A6	Toggles between the **Rotation** and **External axes** groups.
Undo	Undoes the transformation and restores the old point data.
Calculate	Calculates the point transformation and applies it to all selected motion points. If the transformation would cause a point to be situated outside the configured workspace, the point is not transformed.

Only selected points with a complete E6POS definition are copied. This includes, for example, all those that were generated via inline forms during programming. Points without a complete E6POS definition are ignored during the point offset.

10.6.4.3 “Transform - Cartesian Base” window

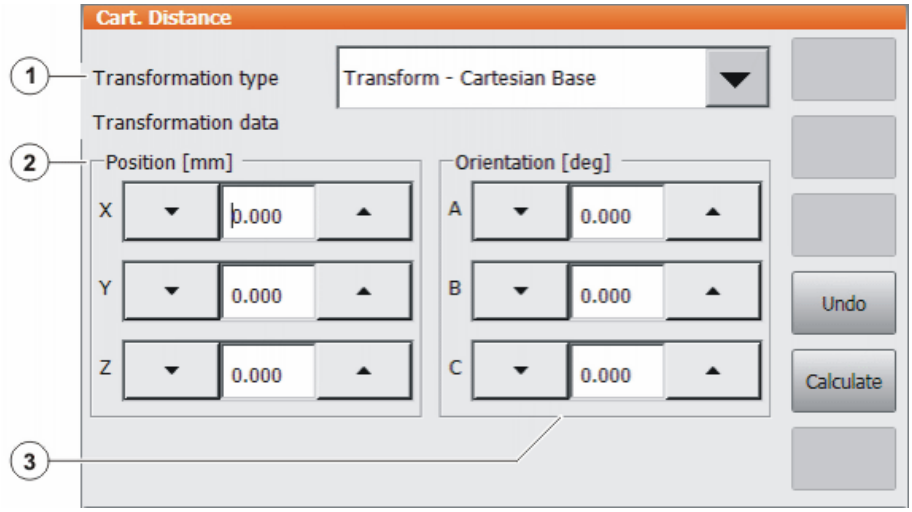


Fig. 10-39: Point transformation - Cartesian

Item	Description
1	Selection of the transformation type
2	**Position** group: input boxes for the point transformation in the X, Y, Z direction • Range of values: Dependent on the configuration of the Cartesian workspaces
3	**Orientation** group: input boxes for the transformation of the A, B, C orientation • Range of values: Dependent on the configuration of the Cartesian workspaces

The following buttons are available:

Button	Description
Undo	Undoes the transformation and restores the old point data.
Calculate	Calculates the point transformation and applies it to all selected motion points. If the transformation would cause a point to be situated outside the configured workspace, the point is not transformed.

Only selected points with a complete E6POS definition are copied. This includes, for example, all those that were generated via inline forms during programming. Points without a complete E6POS definition are ignored during the point offset.

10.7 Global points

10.7.1 Global points – overview

Description

By default, the end points for motions are local points. They are only recognized in the module in which they were created.

An end point can however, depending on the motion type, also be created as a global point. Global points are recognized throughout the robot controller and can be used in every module.

Global points are possible with the following motions:

- SLIN (both in spline block and individually)
- SPTP (both in spline block and individually)
- SPL
- LIN and PTP

HOME positions are already global points.

Global points are NOT possible for:

- Spline block as a whole
- SCIRC and CIRC

In the case of global points, only the motion data are saved globally. Other data are saved locally. In the case of points from option packages, these data could include, for example, program numbers or force specifications.

Names

The names of global points may contain a maximum of 22 characters. The controller internally adds a “g” to the name, with the result that the inline form reaches the usual maximum length of 23 characters.

If the name of an existing global point has been changed, the point is automatically converted into a local point. The controller indicates this via a message.

In the editor, the name of global points is in angle brackets:

```
SLIN <P1> Vel=2 m/s CPDAT3
```

Fig. 10-40: Example SLIN: global point in the editor

Path for Global_Points.dat

The robot controller saves global points in the file Global_Points.dat under KRC:\R1\System. The names of the data sets begin with “g” for “global”.

```
DECL GLOBAL E6POS gXP4={X 1765.00,Y 0.0,Z 1784.00,A
↳ 0.0,B 90.0000,C 0.0,S 2,T 2,E1 0.0,E2 0.0,E3 0.0,E4
↳ 0.0,E5 0.0,E6 0.0}

DECL GLOBAL FDAT gFP4={TOOL_NO 1,BASE_NO 0,
↳ IPO_FRAME #BASE,POINT2[] " "}

DECL GLOBAL PDAT gPPDAT2={VEL 100.000,ACC 100.000,
↳ APO_DIST 100.000,APO_MODE #CPTP,GEAR_JERK 50.0000,
↳ EXAX_IGN 0}
```

Fig. 10-41: Example: global points in Global_Points.dat

If a global point is deleted, it is removed from the SRC file. The point data are retained in Global_Points.dat.

External axes/ RoboTeam

Global points can also be used for external axes and for RoboTeam.

The precondition for this is that the user has the rights for the function group **Teach global points**.

10.7.2 Creating a global point (creating new point or converting local point)

Precondition

- User rights: Function group **Teach global points**

> The function group **Teach global points** is deactivated by default. This means that no user group has the right to teach global points. A user group must therefore first be assigned to the function group if this has not already been done.

- A program is selected.
- T1 mode

Procedure

This procedure can be used to create a new global point. An existing local point can be converted into a global point in the same way.

In the latter case, a message may be displayed. (>>> *"Local > global" Page 434*)

1. Enter the name for the point in the inline form. If an existing point is being converted, modify the name if required.

   Max. length: 22 characters.
2. Touch the arrow next to the point name.

Fig. 10-42: Calling point data (example: SLIN)

3. An option window is opened.
   In **Parameter**, activate the check box next to **Global point**.

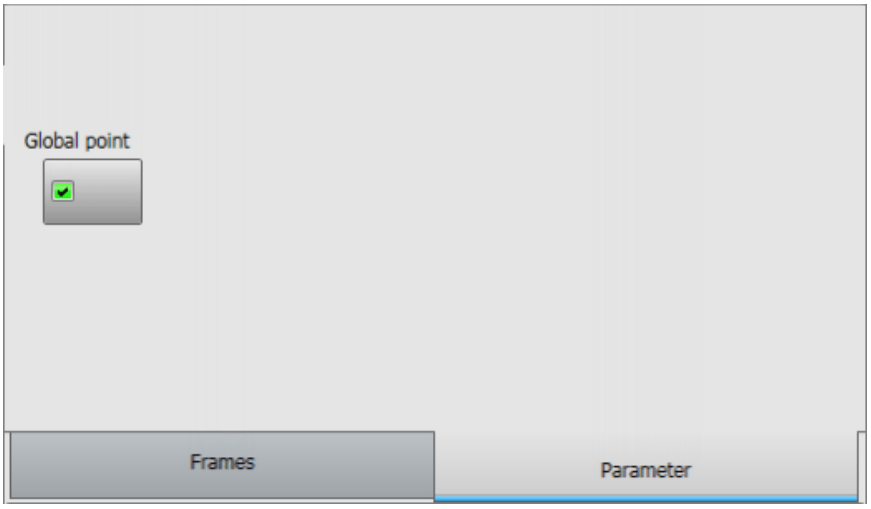


Fig. 10-43: Option window “Parameter”

4. Close the option menu. The point name is now labeled as GLOBAL in the inline form.

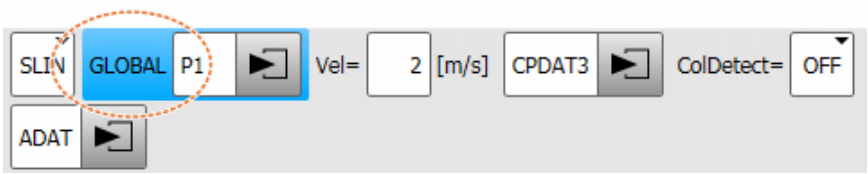


Fig. 10-44: Inline form with global point (using SLIN as an example)

5. Save with **Cmd OK**.

10.7.3 Modifying a global point

As is the case for local points, the inline form for global points can also be opened again and edited (using the **Change** button).

WARNING

If a global point is modified, this affects all modules in which it is used. As a general rule, new or modified programs always need to be tested in T1 mode first. This also applies to modules in which the global point is used.
Failure to observe this precaution may result in injuries or damage to property.

Local > global

Converting a local point to a global point

An existing local point can be converted to a global point. For this, activate the check box next to **Global point**.

Precondition: The point name has a maximum of 22 characters.

The parameters are automatically created in Global_Points.dat as long as the point name does not already exist there. If the name is already there, the following message appears:

Caution: risk of collision when changing global point {0} because it already exists and it could also beused in other modules! Do you want to overwrite existing point "{0}" with current position (TouchUp)? ({1}, {2}, {3}){4}

- Answer **Yes**: The existing point data are overwritten by the new data. This affects all modules in which a global point with this name is used.
- Answer **No**: The existing point data are retained. This affects the module which contains the converted point.

The local data list is not automatically cleaned. The point parameters there are retained – even if the point name is not used again in this module.

Global > local

Converting a global point to a local point

An existing global point can be converted to a local point. For this, deactivate the check box next to **Global point**.

The parameters are automatically created in the local data list as long as the point name does not already exist there. If the name is already there, the following message appears:

Point "{0}" already exists - overwrite? ({1}, {2}, {3})

- Answer **Yes**: The existing point data are overwritten by the new data.
- Answer **No**: The existing point data are retained.

Global_Points.dat is not automatically cleaned. The point parameters there are retained – even if the point name is no longer used in any module.

Further modifications

Reteaching:

Global points can be retaught. The modification affects all modules in which the global point is used. A message with a request for confirmation must be confirmed.

Modifications to tool/base:

Modifications to the tool and/or base affect all modules in which the global point is used. A message with a request for confirmation must be confirmed.

Offsetting/ mirroring

Global points can be offset and mirrored. The precondition for this is that the user has the rights for the function group **Teach global points**.

The modification affects all modules in which the global point is used. A message with a request for confirmation must be confirmed.

10.8 Programming logic instructions

10.8.1 Inputs/outputs

Digital inputs/outputs

The robot controller can manage up to 8192 digital inputs and 8192 digital outputs. 4096 inputs/outputs are available by default.

Analog inputs/outputs

The robot controller can manage 32 analog inputs and 32 analog outputs.

The inputs/outputs are managed via the following system variables:

	Inputs	Outputs
Digital	$IN[1] … $IN[8192]	$OUT[1] … $OUT[8192]
Analog	$ANIN[1] … $ANIN[32]	$ANOUT[1] … $ANOUT[32]

$ANIN[…] indicates the input voltage, adapted to the range between -1.0 and +1.0. The actual voltage depends on the settings of the analog module.

$ANOUT[…] can be used to set an analog voltage. $ANOUT[…] can have values from -1.0 to +1.0 written to it. The voltage actually generated depends on the settings of the analog module. If an attempt is made to set voltages outside the range of values, the robot controller displays the following message: *Limit {Signal name}*

10.8.2 Setting a digital output - OUT

Precondition

- A program is selected.
- Operating mode T1

Procedure

1. Position the cursor in the line after which the logic instruction is to be inserted.
2. Select the menu sequence **Commands** > **Logic** > **OUT** > **OUT**.
3. Set the parameters in the inline form.
   (>>> *10.8.3 "Inline form “OUT”" Page 436*)
4. Save instruction with **Cmd Ok**.

10.8.3 Inline form “OUT”

The instruction sets a digital output.

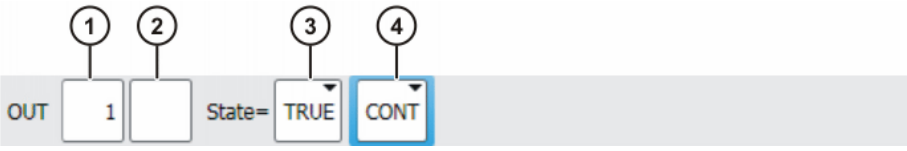


Fig. 10-45: Inline form “OUT”

Item	Description
1	Number of the output
2	If a name exists for the output, this name is displayed. User group “Expert” or higher: A name can be entered by pressing **Long text**. The name is freely selectable.
3	State to which the output is switched • **TRUE** • **FALSE**
4	• **CONT**: Execution in the advance run • **[Empty box]**: Execution with advance run stop

10.8.4 Setting a pulse output - PULSE

Precondition

- A program is selected.
- Operating mode T1

Procedure

1. Position the cursor in the line after which the logic instruction is to be inserted.
2. Select the menu sequence **Commands** > **Logic** > **OUT** > **PULSE**.
3. Set the parameters in the inline form.
   (>>> *10.8.5 "Inline form “PULSE”" Page 437*)
4. Save instruction with **Cmd Ok**.

10.8.5 Inline form “PULSE”

The instruction sets a pulse of a defined length.

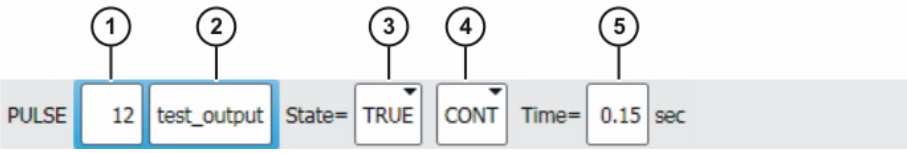


Fig. 10-46: Inline form “PULSE”

Item	Description
1	Number of the output
2	If a name exists for the output, this name is displayed. User group “Expert” or higher: A name can be entered by pressing **Long text**. The name is freely selectable.
3	State to which the output is switched • **TRUE**: “High” level • **FALSE**: “Low” level
4	• **CONT**: Execution in the advance run • **[Empty box]**: Execution with advance run stop
5	Length of the pulse • **0.10 … 3.00 s**

10.8.6 Setting an analog output - ANOUT

Precondition

- A program is selected.
- Operating mode T1

Procedure

1. Position the cursor in the line after which the instruction is to be inserted.

2. Select **Commands** > **Analog output** > **Static** or **Dynamic**.
3. Set the parameters in the inline form.
   (>>> *10.8.7 "Inline form “ANOUT” (static)" Page 438*)
   (>>> *10.8.8 "Inline form “ANOUT” (dynamic)" Page 438*)
4. Save instruction with **Cmd Ok**.

10.8.7 Inline form “ANOUT” (static)

This instruction sets a static analog output. The voltage is set to a fixed level by means of a factor. The actual voltage level depends on the analog module used. For example, a 10 V module with a factor of 0.5 provides a voltage of 5 V.

ANOUT triggers an advance run stop.

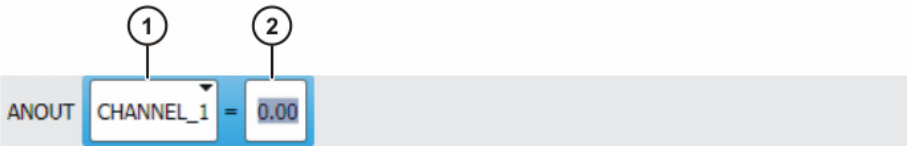


Fig. 10-47: Inline form “ANOUT” (static)

Item	Description
1	Number of the analog output • **CHANNEL_1 … CHANNEL_32**
2	Factor for the voltage • **0 … 1** (intervals: 0.01)

10.8.8 Inline form “ANOUT” (dynamic)

This instruction activates or deactivates a dynamic analog output.

A maximum of 4 dynamic analog outputs can be activated at any one time. ANOUT triggers an advance run stop.

The voltage is determined by a factor. The actual voltage level depends on the following values:

- Velocity or function generator
  For example, a velocity of 1 m/s with a factor of 0.5 results in a voltage of 5 V.
- Offset
  For example, an offset of +0.15 for a voltage of 0.5 V results in a voltage of 6.5 V.

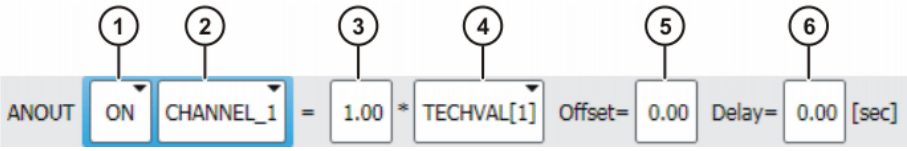


Fig. 10-48: Inline form “ANOUT” (dynamic)

Item	Description
1	Activation or deactivation of the analog output • **ON** • **OFF**
2	Number of the analog output • **CHANNEL_1 … CHANNEL_32**
3	Factor for the voltage • **0 … 10** (intervals: 0.01)
4	• **VEL_ACT**: The voltage is dependent on the velocity. • **TECHVAL[1] … TECHVAL[6]**: The voltage is controlled by a function generator.
5	Value by which the voltage is increased or decreased • **-1 … +1** (intervals: 0.01)
6	Time by which the output signal is delayed (+) or brought forward (-) • **-0.2 … +0.5 s**

10.8.9 Programming a wait time - WAIT

Precondition

- A program is selected.
- Operating mode T1

Procedure

1. Position the cursor in the line after which the logic instruction is to be inserted.
2. Select the menu sequence **Commands** > **Logic** > **WAIT**.
3. Set the parameters in the inline form.
   (>>> *10.8.10 "Inline form “WAIT”" Page 439*)
4. Save instruction with **Cmd Ok**.

10.8.10 Inline form “WAIT”

WAIT can be used to program a wait time. The robot motion is stopped for a programmed time. WAIT always triggers an advance run stop.

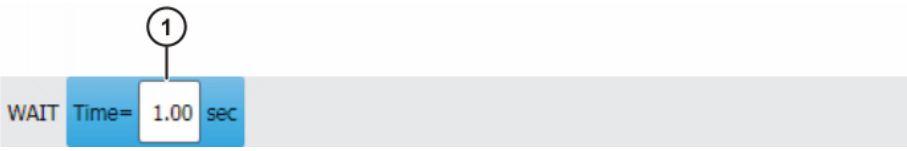


Fig. 10-49: Inline form “WAIT”

Item	Description
1	Wait time • **≥ 0** s

10.8.11 Programming a signal-dependent wait function - WAITFOR

Precondition

- A program is selected.
- Operating mode T1

Procedure

1. Position the cursor in the line after which the logic instruction is to be inserted.
2. Select the menu sequence **Commands** > **Logic** > **WAITFOR**.
3. Set the parameters in the inline form.
   (>>> *10.8.12 "Inline form “WAITFOR”" Page 440*)
4. Save instruction with **Cmd Ok**.

10.8.12 Inline form “WAITFOR”

The instruction sets a signal-dependent wait function.

If required, several signals (maximum 12) can be linked. If a logic operation is added, boxes are displayed in the inline form for the additional signals and links.

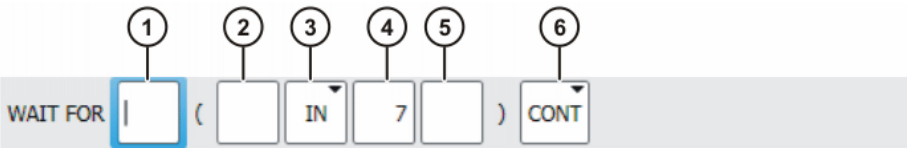


Fig. 10-50: Inline form “WAITFOR”

Item	Description
1	Add external logic operation. The operator is situated between the bracketed expressions. • **AND** • **OR** • **EXOR** Add NOT. • **NOT** • **[blank]** Enter the desired operator by means of the corresponding button.
2	Add internal logic operation. The operator is situated inside a bracketed expression. • **AND** • **OR** • **EXOR** Add NOT. • **NOT** • **[blank]** Enter the desired operator by means of the corresponding button.

Item	Description
3	Signal for which the system is waiting • **IN** • **OUT** • **CYCFLAG** • **TIMER** • **FLAG**
4	Number of the signal
5	If a name exists for the signal, this name is displayed. User group “Expert” or higher: A name can be entered by pressing **Long text**. The name is freely selectable.
6	• **CONT**: Execution in the advance run • **[Empty box]**: Execution with advance run stop

10.8.13 Switching on the path - SYN OUT

Precondition

- A program is selected.
- Operating mode T1

Procedure

1. Position the cursor in the line after which the logic instruction is to be inserted.
2. Select the menu sequence **Commands** > **Logic** > **OUT** > **SYN OUT**.
3. Set the parameters in the inline form.
   (>>> *10.8.14 "Inline form “SYN OUT”, option “START/END”" Page 441*)
   (>>> *10.8.15 "Inline form “SYN OUT”, option “PATH”" Page 444*)
4. Save instruction with **Cmd Ok**.

10.8.14 Inline form “SYN OUT”, option “START/END”

The switching action can be triggered relative to the start or end point of the motion. The switching action can be delayed or brought forward. The motion can be a LIN, CIRC or PTP motion.

Possible applications include:

- Closing or opening the weld gun during spot welding
- Switching the welding current on/off during arc welding
- Starting or stopping the flow of adhesive in bonding or sealing applications.

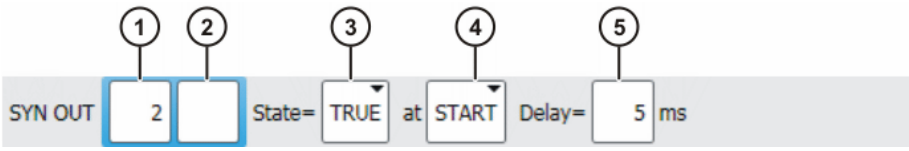


Fig. 10-51: Inline form “SYN OUT”, option “START/END”

Item	Description
1	Number of the output
2	If a name exists for the output, this name is displayed. User group “Expert” or higher: A name can be entered by pressing **Long text**. The name is freely selectable.
3	State to which the output is switched • **TRUE** • **FALSE**
4	Point to which SYN OUT refers: • **START**: Start point of the motion • **END**: End point of the motion
5	Switching action delay • **-1,000 … +1,000 ms** **Note:** The time specification is absolute. In other words, the switching point varies according to the velocity of the robot.

Example 1

Start point and end point are exact positioning points.

```
LIN P1 VEL=0.3m/s CPDAT1
LIN P2 VEL=0.3m/s CPDAT2
SYN OUT 1 '' State= TRUE at START Delay=20ms
SYN OUT 2 '' State= TRUE at END Delay=-20ms
LIN P3 VEL=0.3m/s CPDAT3
LIN P4 VEL=0.3m/s CPDAT4
```

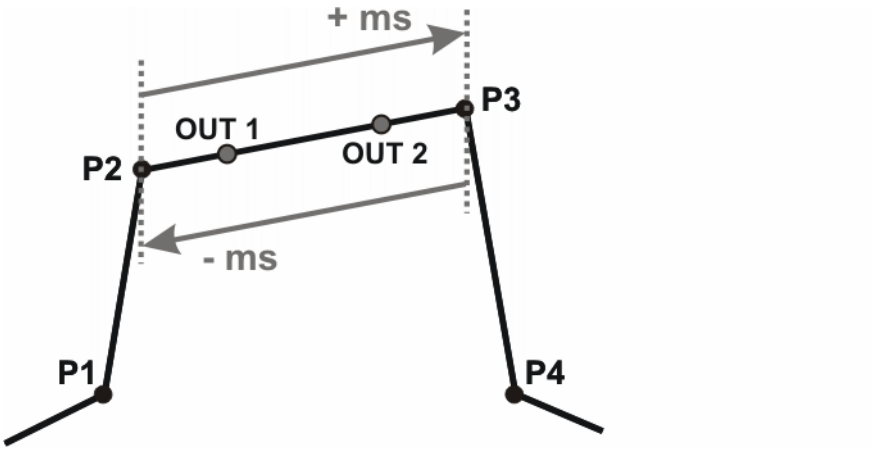


Fig. 10-52

OUT 1 and OUT 2 specify approximate positions at which switching is to occur. The dotted lines indicate the switching limits.

Switching limits:

- START: The switching point can be delayed, at most, as far as exact positioning point P3 (+ ms).
- END: The switching point can be brought forward, at most, as far as exact positioning point P2 (- ms).

If greater values are specified for the delay, the controller automatically switches at the switching limit.

Example 2

Start point is exact positioning point, end point is approximated.

```
LIN P1 VEL=0.3m/s CPDAT1
LIN P2 VEL=0.3m/s CPDAT2
SYN OUT 1 '' State= TRUE at START Delay=20ms
SYN OUT 2 '' State= TRUE at END Delay=-20ms
LIN P3 CONT VEL=0.3m/s CPDAT3
LIN P4 VEL=0.3m/s CPDAT4
```

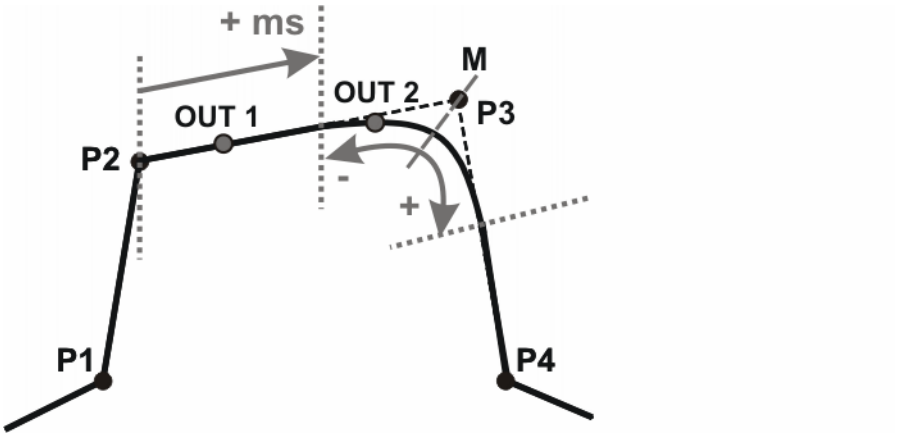


Fig. 10-53

OUT 1 and OUT 2 specify approximate positions at which switching is to occur. The dotted lines indicate the switching limits. M = middle of the approximate positioning range.

Switching limits:

- START: The switching point can be delayed, at most, as far as the start of the approximate positioning range of P3 (+ ms).
- END: The switching point can be brought forward, at most, as far as the start of the approximate positioning range of P3 (-).

  The switching point can be delayed, at most, as far as the end of the approximate positioning range of P3 (+).

If greater values are specified for the delay, the controller automatically switches at the switching limit.

Example 3

Start point and end point are approximated

```
LIN P1 VEL=0.3m/s CPDAT1
LIN P2 CONT VEL=0.3m/s CPDAT2
SYN OUT 1 '' State= TRUE at START Delay=20ms
SYN OUT 2 '' State= TRUE at END Delay=-20ms
LIN P3 CONT VEL=0.3m/s CPDAT3
LIN P4 VEL=0.3m/s CPDAT4
```

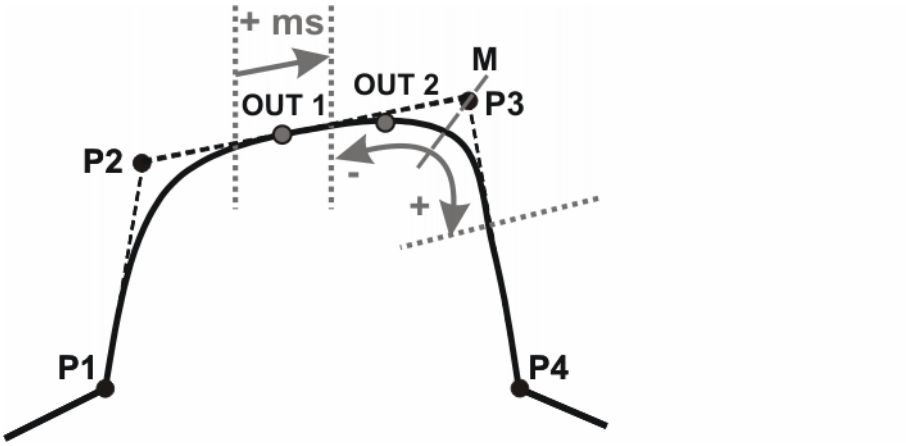


Fig. 10-54

OUT 1 and OUT 2 specify approximate positions at which switching is to occur. The dotted lines indicate the switching limits. M = middle of the approximate positioning range.

Switching limits:

- START: The switching point can be situated, at the earliest, at the end of the approximate positioning range of P2.

  The switching point can be delayed, at most, as far as the start of the approximate positioning range of P3 (+ ms).
- END: The switching point can be brought forward, at most, as far as the start of the approximate positioning range of P3 (-).

  The switching point can be delayed, at most, as far as the end of the approximate positioning range of P3 (+).

If greater values are specified for the delay, the controller automatically switches at the switching limit.

10.8.15 Inline form “SYN OUT”, option “PATH”

The switching action refers to the end point of the motion. The switching action can be shifted in space and delayed or brought forward. The motion can be a LIN or CIRC motion. It must not be a PTP motion.

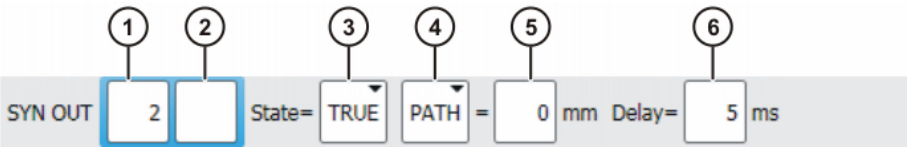


Fig. 10-55: Inline form “SYN OUT”, option “PATH”

Item	Description
1	Number of the output
2	If a name exists for the output, this name is displayed. User group “Expert” or higher: A name can be entered by pressing **Long text**. The name is freely selectable.
3	State to which the output is switched • **TRUE** • **FALSE**

Item	Description
4	• **PATH**: SYN OUT refers to the end point of the motion.
5	This box is only displayed if **PATH** has been selected. Distance from the switching point to the end point • **-2,000 … +2,000 mm**
6	Switching action delay • **-1,000 … +1,000 ms** **Note:** The time specification is absolute. In other words, the switching point varies according to the velocity of the robot.

Example 1

Start point is exact positioning point, end point is approximated.

```
LIN P1 VEL=0.3m/s CPDAT1
SYN OUT 1 '' State= TRUE at START PATH=20mm Delay=-5ms
LIN P2 CONT VEL=0.3m/s CPDAT2
LIN P3 CONT VEL=0.3m/s CPDAT3
LIN P4 VEL=0.3m/s CPDAT4
```

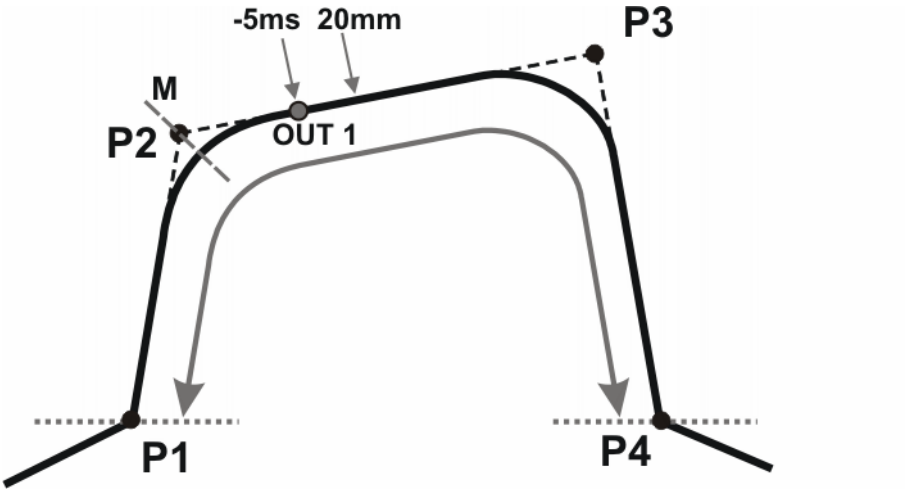


Fig. 10-56

OUT 1 specifies the approximate position at which switching is to occur. The dotted lines indicate the switching limits. M = middle of the approximate positioning range.

Switching limits:

- The switching point can be brought forward, at most, as far as exact positioning point P1.
- The switching point can be delayed, at most, as far as the next exact positioning point P4. If P3 was an exact positioning point, the switching point could be delayed, at most, as far as P3.

If greater values are specified for the shift in space or time, the controller automatically switches at the switching limit.

Example 2

Start point and end point are approximated

```
LIN P1 CONT VEL=0.3m/s CPDAT1
SYN OUT 1 '' State= TRUE at START PATH=20mm Delay=-5ms
LIN P2 CONT VEL=0.3m/s CPDAT2
LIN P3 CONT VEL=0.3m/s CPDAT3
LIN P4 VEL=0.3m/s CPDAT4
```

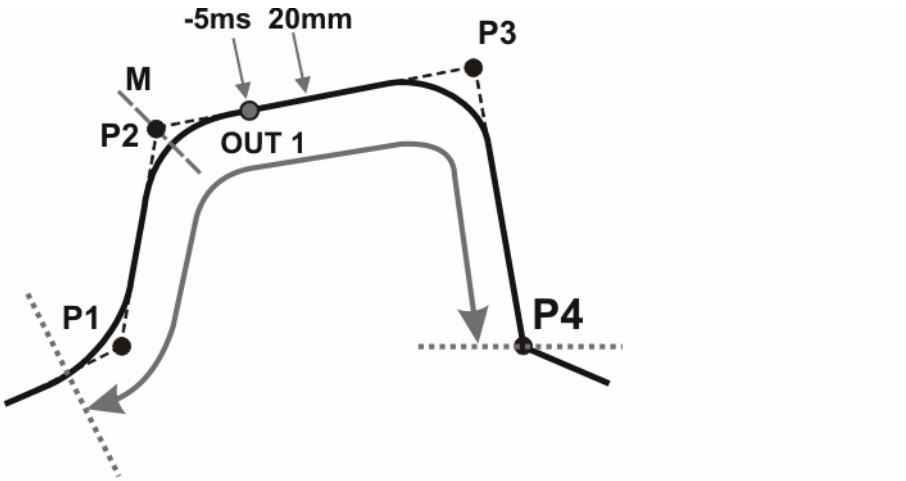


Fig. 10-57

OUT 1 specifies the approximate position at which switching is to occur. The dotted lines indicate the switching limits. M = middle of the approximate positioning range.

Switching limits:

- The switching point can be brought forward, at most, as far as the start of the approximate positioning range of P1.
- The switching point can be delayed, at most, as far as the next exact positioning point P4. If P3 was an exact positioning point, the switching point could be delayed, at most, as far as P3.

If greater values are specified for the shift in space or time, the controller automatically switches at the switching limit.

10.8.16 Setting a pulse on the path - SYN PULSE

Precondition

- A program is selected.
- Operating mode T1

Procedure

1. Position the cursor in the line after which the logic instruction is to be inserted.
2. Select the menu sequence **Commands** > **Logic** > **OUT** > **SYN PULSE**.
3. Set the parameters in the inline form.
   (>>> *10.8.17 "Inline form “SYN PULSE”" Page 447*)
4. Save instruction with **Cmd Ok**.

10.8.17 Inline form “SYN PULSE”

SYN PULSE can be used to trigger a pulse at the start or end point of the motion. The pulse can be shifted in time and/or space, i.e. it does not have to be triggered exactly at the point, but can also be triggered before or after it.

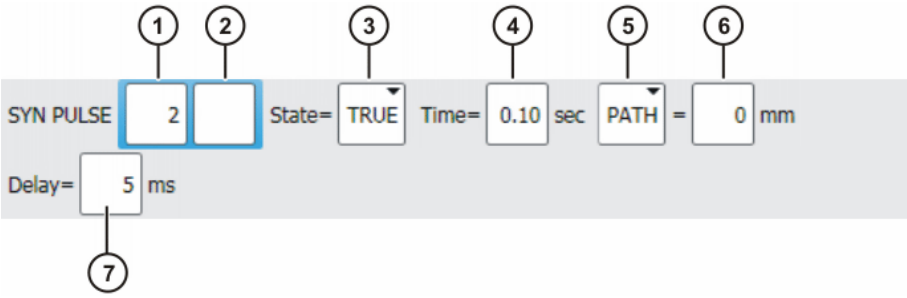


Fig. 10-58: Inline form “SYN PULSE”

Item	Description
1	Number of the output
2	If a name exists for the output, this name is displayed. User group “Expert” or higher: A name can be entered by pressing **Long text**. The name is freely selectable.
3	State to which the output is switched • **TRUE** • **FALSE**
4	Duration of the pulse • **0.1 … 3 s**
5	Point to which SYN PULSE refers: • **START**: Start point of the motion • **END**: End point of the motion See SYN OUT for examples and switching limits. (>>> *10.8.14 "Inline form “SYN OUT”, option “START/END”" Page 441*) • **PATH**: SYN PULSE refers to the end point. An offset in space is also possible. See SYN OUT for examples and switching limits. (>>> *10.8.15 "Inline form “SYN OUT”, option “PATH”" Page 444*)
6	Distance from the switching point to the end point • **-2,000 … +2,000 mm** This box is only displayed if **PATH** has been selected.
7	Switching action delay • **-1,000 … +1,000 ms** **Note:** The time specification is absolute. The switching point varies according to the velocity of the robot.

10.8.18 Modifying a logic instruction

Precondition

- A program is selected.
- Operating mode T1

Procedure

1. Position the cursor in the line containing the instruction that is to be changed.
2. Press **Change**. The inline form for this instruction is opened.
3. Change the parameters.
4. Save changes by pressing **Cmd Ok**.

11 KRL programming

11.1 Instructions for programming

NOTICE

When programming motions, it must be ensured that the energy supply system is not wound up or damaged during program execution.

NOTICE

In the case of programs with the following axis motions or positions, the film of lubricant on the gear units of the axes may break down:

- Motions <3°
- Oscillating motions
- Areas of gear units permanently facing upwards

It must be ensured that the gear units have a sufficient supply of oil. For this, in the case of oscillating motions or short motions (<3°), programming must be carried out in such a way that the affected axes regularly move more than 40° (e.g. once per cycle).
In the case of areas of gear units permanently facing upwards, sufficient oil supply must be achieved by programming re-orientations of the in-line wrist. In this way, the oil can reach all areas of the gear units by means of gravity. Required frequency of re-orientations:

- With low loads (gear unit temperature <+35 °C): daily
- With medium loads (gear unit temperature +35 °C to 55 °C): hourly
- With heavy loads (gear unit temperature >+55 °C): every 10 minutes

Failure to observe this precaution may result in damage to the gear units.

11.2 Symbols and fonts

The following symbols and fonts are used in the syntax descriptions:

Syntax element	Representation
KRL code	• Courier font • Upper-case letters Examples: `GLOBAL`; `ANIN ON`; `OFFSET`
Elements that must be replaced by program-specific entries	• Italics • Upper/lower-case letters Examples: *Distance*; *Time*; *Format*
Optional elements	• In angle brackets Example: <`STEP` *Step_width*>
Elements that are mutually exclusive	• Separated by the “|” symbol Example: `IN`|`OUT`

11.3 Important KRL terms

11.3.1 SRC files and DAT files

A KRL program generally consists of an **SRC file** and a **DAT file** of the same name.

- SRC file: contains the program code.
- DAT file: contains permanent data and point coordinates. The DAT file is also called a **data list**.

The SRC file and the associated DAT file together are called a **module**.

Depending on the user group, programs in the Navigator are displayed as modules or individual files:

- User group “User”

  A program is displayed as a module. The SRC file and the DAT file exist in the background. They are not visible for the user and cannot be edited individually.
- User group “Expert” or higher

  By default, the SRC file and the DAT file are displayed individually. They can be edited individually.

11.3.2 Naming conventions and keywords

Names

Examples of names in KRL: variable names, program names, point names

- Names in KRL can have a maximum length of 24 characters.

  In some cases, less than 24 characters are allowed, e.g. a maximum of 23 characters in inline forms.
- Names in KRL can consist of letters (A-Z), numbers (0-9) and the signs “_” and “$”.
- Names in KRL must not begin with a number.
- Names in KRL must not be keywords.

Other restrictions may apply in the case of inline forms in technology packages.

> The names of all system variables begin with the “$” sign. To avoid confusion, do not begin the names of user-defined variables with this sign.

Keywords

Keywords are sequences of letters having a fixed meaning. They must not be used in programs in any way other than with this meaning. No distinction is made between uppercase and lowercase letters. A keyword remains valid irrespective of the way in which it is written.

Example: The sequence of letters CASE is an integral part of the KRL syntax SWITCH … CASE … ENDSWITCH. For this reason, CASE must not be used in any other way, e.g. as a variable name.

The system distinguishes between reserved and non-reserved keywords:

- Reserved keywords

  These may only be used with their defined meaning.

- Non-reserved keywords

  With non-reserved keywords, the meaning is restricted to a particular context. Outside of this context, a non-reserved keyword is interpreted by the compiler as a name.

> In practice, it is not helpful to distinguish between reserved and non-reserved keywords. To avoid error messages or compiler problems, keywords are thus never used other than with their defined meaning.

Overview of important keywords:

All elements of the KRL syntax described in this documentation that are not program-specific are keywords.

The following important keywords are worth a particular mention:

AXIS	ENDFCT
BOOL	ENDFOR
CHAR	ENDIF
CAST_FROM	ENDLOOP
CAST_TO	ENDSWITCH
CCLOSE	ENDWHILE
CHANNEL	EXT
CIOCTL	EXTFCT
CONFIRM	FALSE
CONST	FRAME
COPEN	GLOBAL
CREAD	INT
CWRITE	MAXIMUM
DEF	MINIMUM
DEFAULT	POS
DEFDAT	PRIO
DEFFCT	PUBLIC
E6AXIS	SREAD
E6POS	SWRITE
END	REAL
ENDDAT	TRUE

11.3.3 Data types

Overview

There are 2 kinds of data types:

- User-defined data types

  User-defined data types are always derived from the data types ENUM or STRUC.
- Predefined data types, e.g.:
  - Simple data types
  - Data types for motion programming

The following simple data types are predefined:

Data type	Keyword	Description
Integer	INT	Integer • **$-2^{31}-1$ … $2^{31}-1$** Examples: 1; 32; 345
Real	REAL	Floating-point number • **+1.1E-38 … +3.4E+38** Examples: 1.43; 38.50; 300.25
Boolean	BOOL	Logic state • **TRUE** • **FALSE**
Character	CHAR	1 character • **ASCII character** Examples: "A"; "1"; "q"

The following data types for motion programming are predefined:

Structure type AXIS

A1 to A6 are angle values (rotational axes) or translation values (translational axes) for the axis-specific movement of robot axes 1 to 6.

```
STRUC AXIS REAL A1, A2, A3, A4, A5, A6
```

Structure type E6AXIS

E1 to E6 are angle values or translation values of the external axes 7 to 12.

```
STRUC E6AXIS REAL A1, A2, A3, A4, A5, A6, E1, E2, E3, E4,
E5, E6
```

Structure type FRAME

X, Y and Z are space coordinates, while A, B and C are the orientation of the coordinate system.

```
STRUC FRAME REAL X, Y, Z, A, B, C
```

Structure types POS and E6POS

S (Status) and T (Turn) define axis positions unambiguously.

```
STRUC POS REAL X, Y, Z, A, B, C, INT S, T
```

```
STRUC E6POS REAL X, Y, Z, A, B, C, E1, E2, E3, E4, E5, E6,
INT S, T
```

11.3.4 Areas of validity

Local

Data object	Area of validity
Variable Signal	• If the object was defined in an SRC file: It is valid in the program routine in which it was defined, i.e. between DEF and END (main program OR local subprogram). Variables defined in an SRC file are called “runtime variables”. • If the object was defined in a DAT file: It is valid in the SRC file that belongs to the DAT file.
Constant	Valid in the module to which the data list in which the constant was declared belongs.
User-defined data type	• If the data type was defined in an SRC file: it is valid at, or below, the program level in which it was declared. • If the data type was defined in a DAT file: it is valid in the SRC file that belongs to the DAT file.
Subprogram	Valid in the main program of the shared SRC file.
Function	Valid in the main program of the shared SRC file.
Interrupt	Valid at, or below, the programming level in which it was declared.

Global

Always globally valid:

- The first program in an SRC file. By default, it bears the name of the SRC file.
- Predefined data types
- KRL system variables
- Variables and signals defined in $CONFIG.DAT

The data objects named under “local” can be made available globally.

- (>>> *11.3.4.1 "Making subprograms, functions and interrupts available globally" Page 453*)
- (>>> *11.3.4.2 "Making variables, constants, signals and user data types available globally" Page 454*)

If there are local and global objects with the same name, the compiler uses the local object within its area of validity.

11.3.4.1 Making subprograms, functions and interrupts available globally

Use the keyword GLOBAL in the declaration.

Example of subprogram:

```
...
END
-------------------------------
GLOBAL DEF MY_SUBPROG
...
```

Example of function:

```
...
END
-------------------------------
```

```
GLOBAL DEFFCT INT MY_FCT(my_var:IN)
...
```

Example of interrupt:

```
GLOBAL INTERRUPT DECL 23 WHEN $IN[12]==TRUE DO UP1(20,VALUE)
```

11.3.4.2 Making variables, constants, signals and user data types available globally

Variables, signals and user-defined data types can be made available globally via a data list or $CONFIG.DAT.

Constants must always be declared and, at the same time, initialized in a data list. For this reason, they can only be made available globally via a data list.

Data list

Making the object available globally via a data list:

1. Insert the keyword PUBLIC into the program header of the data list:

```
DEFDAT MY_PROG PUBLIC
```

2. Use the keyword GLOBAL in the declaration.

   Example (declaration of a variable):

```
DEFDAT MY_PROG PUBLIC
EXTERNAL DECLARATIONS
DECL GLOBAL INT counter
...
ENDDAT
```

GLOBAL can only be used for variables, signals and user-defined data types if they have been declared in a data list.

PUBLIC is used exclusively for the purpose described here, i.e. together with GLOBAL in data lists for making specific data objects available globally. PUBLIC on its own has no effect.

$CONFIG.DAT

- Declare the object in the USER GLOBALS section in $CONFIG.DAT.

The keyword GLOBAL is not required here, nor can it be used here.

Restriction

Data types defined in a data list using the keyword GLOBAL must not be used in $CONFIG.DAT.

Example:

In DEFDAT PROG(), the enumeration type SWITCH_TYP has been defined with the keyword GLOBAL:

```
DEFDAT PROG()
GLOBAL ENUM SWITCH_TYP ON, OFF
...
```

If this data type is used in $CONFIG.DAT, the compiler signals the error "*Type unknown: *** DECL SWITCH_TYP MY_VAR*".

```
DEFDAT $CONFIG
DECL SWITCH_TYP MY_VAR
```

```
...
```

11.3.5 Constants

The value of a constant can no longer be modified during program execution after initialization. Constants can be used to prevent a value from being changed accidentally during program execution.

Constants must be declared and, at the same time, initialized in a data list. The data type must be preceded by the keyword CONST.

`DECL <GLOBAL> CONST` *Data type Variable name = Value*

11.4 Variables and declarations

11.4.1 DECL: declaring variables, arrays and constants

Syntax

Declaration of variables

Declaration of variables in programs:

`<DECL>` *Data type Name1 <, ..., NameN>*

Declaration of variables in data lists:

`<DECL> <GLOBAL>` *Data type Name1 <, ..., NameN>*

Declaration of variables in data lists with simultaneous initialization:

`<DECL> <GLOBAL>` *Data type Name = Value*

In the case of declaration with simultaneous initialization, a separate DECL declaration is required for each variable. It is not possible to declare and initialize several variables with a single DECL declaration.

Declaration of arrays

Declaration of arrays in programs:

`<DECL>` *Data type Name1 [Dimension1 <, ..., Dimension3>] <, ..., NameN [DimensionN1 <,..., DimensionN3>] >*

Declaration of arrays in data lists:

`<DECL> <GLOBAL>` *Data type Name1 [Dimension1 <, ..., Dimension3>] <, ..., NameN [DimensionN1 <,..., DimensionN3>] >*

For the declaration of arrays or constant arrays in data lists with simultaneous initialization:

- It is not permissible to declare and initialize in a single line. The initialization must, however, follow directly after the line containing the declaration. There must be no lines, including blank lines, in between.
- If several elements of an array are initialized, the elements must be specified in ascending sequence of the array index (starting from the right-hand array index).
- If the same character string is to be assigned to all of the elements of an array of type CHAR as a default setting, it is not necessary to initialize each array element individually. The right-hand array index is omitted. (No index is written for a one-dimensional array index.)

Declaration of arrays in data lists with simultaneous initialization:

`<DECL> <GLOBAL>` *Data type Name [Dimension1 <,..., Dimension3>]*
Name [1 <, 1, 1>] = Value1
<Name [1 <, 1, 2>] = Value2>

. . .
Name [Dimension1 <, Dimension2, Dimension3>] = ValueN

Declaration of constant arrays in data lists with simultaneous initialization:

`DECL <GLOBAL> CONST` *Data type Name [Dimension1 <,..., Dimension3>]*
Name [1 <, 1, 1>] = Value1
<Name [1 <, 1, 2>] = Value2>
. . .
Name [Dimension1 <, Dimension2, Dimension3>] = ValueN

Explanation of the syntax

Element	Description
`DECL`	DECL can be omitted if *Data type* is a predefined data type. If *Data type* is a user-defined data type, then DECL is obligatory.
`GLOBAL`	(>>> *11.3.4 "Areas of validity" Page 453*)
`CONST`	The keyword CONST must only be used in data lists.
Data type	Specification of the desired data type
Name	Name of the object (variable, array or constant) that is being declared.
Dimension	Type: INT *Dimension* defines the number of array elements for the dimension in question. Arrays have a minimum of 1 and a maximum of 3 dimensions.
Value	The data type of *Value* must be compatible with *Data type*, but not necessarily identical. If the data types are compatible, the system automatically matches them.

Example 1

Declarations with predefined data types. The keyword DECL can also be omitted.

```
DECL INT X
DECL INT X1, X2
DECL REAL ARRAY_A[7], ARRAY_B[5], A
```

Example 2

Declarations of arrays with simultaneous initialization (only possible in data lists).

```
INT A[7]
A[1]=27
A[2]=313
A[6]=11
CHAR TEXT1[80]
TEXT1[]="message"
CHAR TEXT2[2,80]
TEXT2[1,]="first message"
TEXT2[2,]="second message"
```

11.4.2 ENUM: defining an enumeration type

Description

Definition of an enumeration type (= ENUM data type)

Syntax

`<GLOBAL> ENUM` *NameEnumType Constant1*`<,  ...,  `*ConstantN*`>`

Explanation of the syntax

Element	Description
GLOBAL	(>>> *11.3.4 "Areas of validity" Page 453*) **Note:** Data types defined using the keyword GLOBAL must not be used in $CONFIG.DAT.
NameEnum-Type	Name of the new enumeration type. Recommendation: For user-defined data types, assign names ending in _TYPE, to distinguish them from variable names.
Constant	The constants are the values that a variable of the enumeration type can take. Each constant may only occur once in the definition of the enumeration type.

Example 1

Definition of an enumeration type with the name COUNTRY_TYPE.

```
ENUM COUNTRY_TYP SWITZERLAND, AUSTRIA, ITALY, FRANCE
```

Declaration of a variable of type COUNTRY_TYPE:

```
DECL COUNTRY_TYP MYCOUNTRY
```

Initialization of the variable of type COUNTRY_TYPE:

```
MYCOUNTRY = #AUSTRIA
```

Example 2

An enumeration type with the name SWITCH_TYPE and the constants ON and OFF is defined.

```
DEF PROG()
ENUM SWITCH_TYP ON, OFF
DECL SWITCH_TYP GLUE
    IF A>10 THEN
        GLUE=#ON
    ELSE
        GLUE=#OFF
    ENDIF
END
```

Restriction

Data types defined in a data list using the keyword GLOBAL must not be used in $CONFIG.DAT.

Example:

In DEFDAT PROG(), the enumeration type SWITCH_TYP has been defined with the keyword GLOBAL:

```
DEFDAT PROG()
GLOBAL ENUM SWITCH_TYP ON, OFF
...
```

If this data type is used in $CONFIG.DAT, the compiler signals the error "*Type unknown: *** DECL SWITCH_TYP MY_VAR*".

```
DEFDAT $CONFIG
DECL SWITCH_TYP MY_VAR
...
```

11.4.3 STRUC: defining a structure type

Description

Definition of a structure type (= STRUC data type). Several data types are combined to form a new data type.

Syntax

`<GLOBAL> STRUC` *Name structure type Data type1 Component1A*<, *Component1B*, ...> < , *Data type2 Component2A*<, *Component2B*, ...>>

Explanation of the syntax

Element	Description
GLOBAL	(>>> *11.3.4 "Areas of validity" Page 453*) **Note:** Data types defined using the keyword GLOBAL must not be used in $CONFIG.DAT.
Name structure type	Name of the new structure type. The names of user-defined data types should end in _TYPE, to distinguish them from variable names.
Data type	TYPE: Any data type Structure types are also permissible as data types.
Component	Name of the component. It may only be used once in the structure type. Arrays can only be used as components of a structure type if they have the type CHAR and are one-dimensional. In this case, the array limit follows the name of the array in square brackets in the definition of the structure type.

Value assignment

There are 2 ways of assigning values to variables based on a STRUC data type:

- Assignment of values to several components of a variable: with an **aggregate**
- Assignment of a value to a single component of a variable: with the **point separator**

Information regarding the aggregate:

- The values of an aggregate can be simple constants or themselves aggregates; they may not, however, be variables (see also Example 3).

- Not all components of the structure have to be specified in an aggregate.
- The components do not need to be specified in the order in which they have been defined.
- Each component may only be contained once in an aggregate.
- The name of the structure type can be specified at the beginning of an aggregate, separated by a colon.

Example 1

Definition of a structure type CAR_TYPE with the components AIR_COND, YEAR and PRICE.

```
STRUC CAR_TYP BOOL AIR_COND, INT YEAR, REAL PRICE
```

Declaration of a variable of type CAR_TYPE:

```
DECL CAR_TYP MYCAR
```

Initialization of the variable MYCAR of type CAR_TYPE with an **aggregate**:

```
MYCAR = {CAR_TYP: PRICE 15000, AIR_COND TRUE, YEAR 2003}
```

A variable based on a structure type does not have to be initialized with an aggregate. It is also possible to initialize the components individually with the point separator.

Modification of an individual component using the **point separator**:

```
MYCAR.AIR_COND = FALSE
```

Example 2

Definition of a structure type S_TYPE with the component NUMBER of data type REAL and of the array component TEXT[80] of data type CHAR.

```
STRUC S_TYP REAL NUMBER, CHAR TEXT[80]
```

Example 3

Example of aggregates as values of an aggregate:

```
STRUC INNER_TYP INT A, B, C
STRUC OUTER_TYP INNER_TYP Q, R
DECL OUTER_TYP MYVAR
...
MYVAR = {Q {A 1, B 4}, R {A 3, C 2}}
```

11.5 Motion programming: PTP, LIN, CIRC

11.5.1 PTP: motion programming

Description

Executes a point-to-point motion to the end point. The coordinates of the end point are absolute.

Syntax

`PTP` *End point* <*Approximation*>

Explanation of the syntax

Element	Description
End point	Type: POS, E6POS, AXIS, E6AXIS, FRAME The end point can be specified in Cartesian or axis-specific coordinates. Cartesian coordinates refer to the BASE coordinate system. If not all components of the end point are specified, the controller takes the values of the previous position for the missing components.
Approximation (blending)	*Approximation* causes the end point to be approximated. At the same time, this parameter defines the earliest point at which the approximate positioning can begin. (>>> *11.5.7 "Approximation parameters for PTP, LIN CIRC and …_REL" Page 466*)

Example 1

End point specified in Cartesian coordinates.

```
PTP {X 12.3,Y 100.0,Z 50,A 9.2,B 50,C 0,S 'B010',T 'B1010'}
```

Example 2

End point specified in axis-specific coordinates. The end point is approximated.

```
PTP {A1 10,A2 -80.6,A3 -50,A4 0,A5 14.2, A6 0} C_DIS
```

Example 3

End point specified with only 2 components. For the rest of the components, the controller takes the values of the previous position.

```
PTP {Z 500,X 123.6}
```

11.5.2 LIN: motion programming

Description

LIN executes a linear motion to the end point. The coordinates of the end point are absolute.

Syntax

`LIN` *End point* <*Approximation*>

Explanation of the syntax

Element	Description
End point	Type: POS, E6POS, FRAME If not all components of the end point are specified, the controller takes the values of the previous position for the missing components. The Status and Turn specifications for an end point of type POS or E6POS are disregarded. The coordinates refer to the BASE coordinate system.
Approximation (blending)	*Approximation* causes the end point to be approximated. At the same time, this parameter defines the earliest point at which the approximate positioning can begin. (>>> *11.5.7 "Approximation parameters for PTP, LIN CIRC and …_REL" Page 466*)

Example

End point with two components. For the rest of the components, the controller takes the values of the previous position.

```
LIN {Z 500,X 123.6}
```

11.5.3 CIRC: motion programming

Description

CIRC executes a circular motion. An auxiliary point and an end point must be specified in order for the controller to be able to calculate the circular motion.

The coordinates of the auxiliary point and end point are absolute.

Syntax

`CIRC` *Auxiliary point, End point*<, `CA` *Circular angle*> <*Approximation*>

Explanation of the syntax

Element	Description
Auxiliary point	Type: POS, E6POS, FRAME If not all components of the auxiliary point are specified, the controller takes the values of the previous position for the missing components. The orientation angles and the Status and Turn specifications for an auxiliary point are always disregarded. The auxiliary point cannot be approximated. The motion always stops exactly at this point. The coordinates refer to the BASE coordinate system.
End point	Type: POS, E6POS, FRAME If not all components of the end point are specified, the controller takes the values of the previous position for the missing components. The Status and Turn specifications for an end point of type POS or E6POS are disregarded. The coordinates refer to the BASE coordinate system.
Circular angle	Unit: Degrees; without restriction (>>> *9.9 "Circular angle" Page 383*)
Approximation (blending)	*Approximation* causes the end point to be approximated. At the same time, this parameter defines the earliest point at which the approximate positioning can begin. (>>> *11.5.7 "Approximation parameters for PTP, LIN CIRC and …_REL" Page 466*)

Example

The end point of the circular motion is defined by a circular angle of 260°. The end point is approximated.

```
CIRC {X 5,Y 0, Z 9.2},{X 12.3,Y 0,Z -5.3,A 9.2,B -5,C 20},
CA 260 C_ORI
```

11.5.4 PTP_REL: relative motion programming

Description

Executes a point-to-point motion to the end point. The coordinates of the end point are relative to the current position.

> A REL statement always refers to the current position of the robot. For this reason, if a REL motion is interrupted, the robot executes the entire REL motion again, starting from the position at which it was interrupted.

For information about the response of the robot controller in the case of infinitely rotating axes: (>>> *11.5.8 "REL motions for infinitely rotating axes" Page 467*)

Syntax

`PTP_REL` *End point* <*Approximation*> <#BASE|#TOOL>

Explanation of the syntax

Element	Description
End point	Type: POS, E6POS, AXIS, E6AXIS The end point can be specified in Cartesian or axis-specific coordinates. The controller interprets the coordinates as relative to the current position. Cartesian coordinates refer to the BASE coordinate system. If not all components of the end point are specified, the controller sets the value of the missing components to 0. In other words, the absolute values of these components remain unchanged.
Approximation	*Approximation* causes the end point to be approximated. At the same time, this parameter defines the earliest point at which the approximate positioning can begin. (>>> *11.5.7 "Approximation parameters for PTP, LIN CIRC and ..._REL" Page 466*)
`#BASE`, `#TOOL`	Only permissible if the end point was specified in Cartesian coordinates. • #BASE (default): The coordinates of this end point refer to the coordinate system that belongs to the physical base. • #TOOL: The coordinates of this end point refer to the coordinate system that belongs to the physical tool. $IPO_MODE has no influence on the meaning of #BASE and #TOOL.

Example 1

Axis 2 is moved 30 degrees in a negative direction. None of the other axes moves.

```
PTP_REL {A2 -30}
```

Example 2

The robot moves 100 mm in the X direction and 200 mm in the negative Z direction from the current position. Y, A, B, C and S remain constant. T is calculated in relation to the shortest path.

```
PTP_REL {X 100,Z -200}
```

11.5.5 LIN_REL: relative motion programming

Description

LIN_REL executes a linear motion to the end point. The coordinates of the end point are relative to the current position.

> A REL statement always refers to the current position of the robot. For this reason, if a REL motion is interrupted, the robot executes the entire REL motion again, starting from the position at which it was interrupted.

For information about the response of the robot controller in the case of infinitely rotating axes: (>>> *11.5.8 "REL motions for infinitely rotating axes" Page 467*)

Syntax

`LIN_REL` *End point* <*Approximation*> `<#BASE|#TOOL>`

Explanation of the syntax

Element	Description
End point	Type: POS, E6POS, FRAME The end point must be specified in Cartesian coordinates. The controller interprets the coordinates as relative to the current position. The coordinates can refer to the BASE or TOOL coordinate system. If not all components of the end point are specified, the controller sets the value of the missing components to 0. In other words, the absolute values of these components remain unchanged. The Status and Turn specifications for an end point of type POS or E6POS are disregarded.
Approximation (blending)	*Approximation* causes the end point to be approximated. At the same time, this parameter defines the earliest point at which the approximate positioning can begin. (>>> *11.5.7 "Approximation parameters for PTP, LIN CIRC and …_REL" Page 466*)
`#BASE`, `#TOOL`	• #BASE (default): The coordinates of this end point refer to the coordinate system that belongs to the physical base. • #TOOL: The coordinates of this end point refer to the coordinate system that belongs to the physical tool. $IPO_MODE has no influence on the meaning of #BASE and #TOOL.

Information about $APO is contained in the **System Variables** documentation.

Examples

Example 1:

The TCP moves 100 mm in the X direction and 200 mm in the negative Z direction from the current position in the BASE coordinate system. Y, A, B, C and S remain constant. T is determined by the motion.

```
LIN_REL {X 100,Z -200}
```

Example 2:

The TCP moves 100 mm from the current position in the negative X direction in the TOOL coordinate system. Y, Z, A, B, C and S remain constant. T is determined by the motion.

This example is suitable for moving the tool backwards against the tool direction. The precondition is that the tool direction has been calibrated along the X axis.

```
LIN_REL {X -100} #TOOL
```

11.5.6 CIRC_REL: relative motion programming

Description

CIRC_REL executes a circular motion. An auxiliary point and an end point must be specified in order for the controller to be able to calculate the circular motion. The coordinates of the auxiliary point and end point are relative to the current position.

> A REL statement always refers to the current position of the robot. For this reason, if a REL motion is interrupted, the robot executes the entire REL motion again, starting from the position at which it was interrupted.

For information about the response of the robot controller in the case of infinitely rotating axes: (>>> *11.5.8 "REL motions for infinitely rotating axes" Page 467*)

Syntax

`CIRC_REL` *Auxiliary point, End point*`<,` `CA` *Circular angle*`>` `<`*Approximation*`>` `<#BASE|#TOOL>`

Explanation of the syntax

Element	Description
Auxiliary point	Type: POS, E6POS, FRAME The auxiliary point must be specified in Cartesian coordinates. The controller interprets the coordinates as relative to the current position. The coordinates refer to the BASE coordinate system. If $ORI_TYPE, Status and/or Turn are specified, these specifications are ignored. If not all components of the auxiliary point are specified, the controller sets the value of the missing components to 0. In other words, the absolute values of these components remain unchanged. The orientation angles and the Status and Turn specifications for an auxiliary point are disregarded. The auxiliary point cannot be approximated. The motion always stops exactly at this point.
End point	Type: POS, E6POS, FRAME The end point must be specified in Cartesian coordinates. The controller interprets the coordinates as relative to the current position. The coordinates refer to the BASE coordinate system. If not all components of the end point are specified, the controller sets the value of the missing components to 0. In other words, the absolute values of these components remain unchanged. The Status and Turn specifications for an end point of type POS or E6POS are disregarded.
Circular angle	Unit: Degrees; without restriction (>>> *9.9 "Circular angle" Page 383*)

Element	Description
Approximation (blending)	*Approximation* causes the end point to be approximated. At the same time, this parameter defines the earliest point at which the approximate positioning can begin. (>>> *11.5.7 "Approximation parameters for PTP, LIN CIRC and …_REL" Page 466*)
`#BASE`, `#TOOL`	• #BASE (default): The coordinates of this end point refer to the coordinate system that belongs to the physical base. • #TOOL: The coordinates of this end point refer to the coordinate system that belongs to the physical tool. $IPO_MODE has no influence on the meaning of #BASE and #TOOL.

Information about $APO is contained in the **System Variables** documentation.

Example

The end point of the circular motion is defined by a circular angle of 500°. The end point is approximated.

```
CIRC_REL {X 100,Y 3.2,Z -20},{Y 50},CA 500 C_VEL
```

11.5.7 Approximation parameters for PTP, LIN CIRC and …_REL

Parameters

Not every parameter can be used in every instruction.

Parameter	Description
C_PTP	Approximation starts, at the earliest, when half the distance between the start point and end point relative to the contour of the motion without approximation has been covered.
C_DIS	Approximation starts, at the earliest, when the distance to the end point falls below the value of $APO.CDIS.
C_ORI	Approximation starts, at the earliest, when the dominant orientation angle falls below the value of $APO.CORI.
C_VEL	Approximation starts, at the earliest, when the velocity in the deceleration phase to the end point falls below the value of $APO.CVEL.

Information about $APO is contained in the **System Variables** documentation.

PTP, PTP_REL

The parameter must be C_PTP or C_DIS for **PTP-PTP** approximation. If a second parameter is specified, the robot controller ignores it.

Two parameters can be specified for **PTP-CP** approximation. Of the two parameters, the one resulting in the smaller approximate positioning radius in the given situation takes effect.

Possible combinations for PTP-CP approximation:

1st parameter → / 2nd parameter ↓	C_PTP	C_DIS
(without)	Possible	Possible
C_DIS	Possible	Not possible!
C_VEL	Possible	Possible
C_ORI	Possible	Possible

Example: PTP-CP approximation

```
PTP XP1 C_PTP C_DIS
LIN XP2
```

The robot controller calculates the approximate positioning radius which would result from each of the two parameters C_PTP and C_DIS under the current conditions (velocity, etc.). Only the smaller of the two radii then actually has an effect. It is the earliest limit at which approximate positioning can begin.

LIN, CIRC, LIN_REL, CIRC_REL

The parameter must be C_DIS, C_VEL or C_ORI for **CP-CP** approximation. If a second parameter is specified, the robot controller ignores it.

Two parameters can be specified for **CP-PTP** approximation. Of the two parameters, the one resulting in the smaller approximate positioning radius in the given situation takes effect.

Possible combinations for CP-PTP approximation:

1st parameter → / 2nd parameter ↓	C_DIS	C_VEL	C_ORI
(without)	Possible	Possible	Possible
C_PTP	Possible	Possible	Possible
C_DIS	Possible	Possible	Possible

11.5.8 REL motions for infinitely rotating axes

Description

Motion	Description
SPTP_REL	The end position is calculated directly by adding the start position to the value specified in the REL statement. The overall length of the resulting path is irrelevant.
For external axes only: SLIN_REL, SCIRC_REL, SPL_REL	
PTP_REL	The axis only moves to positions with the following interval: • From “Start position minus 180°” • To “Start position plus 180°” If the addition of the start position and the value specified in the REL statement gives a value outside this interval, the actual end position is calculated as the difference of the value from 360° or a multiple of 360°.
For external axes only: LIN_REL, CIRC_REL	

Examples

Let A6 and E1 be infinitely rotating axes with the start position 120°.

Let the position at X be = 1500 mm.

Example 1:

Statement	End position
SPTP_REL {A6 330}	A6 = 450°
PTP_REL {A6 330}	A6 = 90°

Explanation of end position of PTP_REL:

The permissible interval is from -60° to 300°. The position 450° is outside this interval and is thus not addressed.

The end position must be within the interval AND be calculated as follows:

- 450° ± (x * 360°)

The end position that meets these criteria is 90°.

450° - (1 * 360°) = 90°

Example 2:

Statement	End position
SPTP_REL {A6 550}	A6 = 670°
PTP_REL {A6 550}	A6 = -50°

Example 3:

Statement	End position
SPTP_REL {E1 950}	E1 = 1070°
PTP_REL {E1 950}	E1 = -10°

The statements do not contain any specification of the robot position. This implicitly corresponds to: {X 0, Y 0, Z 0, A 0, B 0, C 0}

The Cartesian robot position thus remains unchanged in both cases.

Example 4:

Statement	End position
SPTP_REL {A6 0, E1 950}	A6 = 120°, E1 = 1070°
PTP_REL {A6 0, E1 950}	A6 = 120°, E1 = -10°

The robot position, if not specified, is implicitly Cartesian, as explained in example 3.

If, however, the axis-specific robot position and not the Cartesian position is to remain unchanged, a zero motion must be specified explicitly for at least one robot axis, as illustrated here in example 4.

Example 5:

Statement	End position
SLIN_REL {X 300, E1 880}	X = 1800 mm, E1 = 1000°
LIN_REL {X 300, E1 880}	X = 1800 mm, E1 = 280°

External axis motions are always axis-specific. They are thus specified in degrees, even in these statements that only allow Cartesian coordinates for robot positions.

11.6 Motion programming: spline

11.6.1 SPLINE ... ENDSPLINE: programming a CP spline block

Description

SPLINE … ENDSPLINE defines a CP spline block. A CP spline block may contain:

- SLIN, SCIRC and SPL segments (number limited only by the memory capacity)
- PATH trigger
- 1 time block (TIME_BLOCK …)
  or 1 constant velocity range (CONST_VEL …)
- STOP WHEN PATH
- Comments
- Blank lines

The block must not include any other instructions, e.g. variable assignments or logic statements.

> The start point of a spline block is the last point before the spline block.
> The end point of a spline block is the last point in the spline block.
> A spline block does not trigger an advance run stop.

Syntax

`SPLINE` < `WITH` *SysVar1* = *Value1* <, *SysVar2* = *Value2*, … > >
Segment1
…
<*SegmentN*>
`ENDSPLINE` <`C_SPL`>

Explanation of the syntax

Element	Description
SysVar	(>>> *11.6.11 "System variables for WITH" Page 480*)
Value	Value assignment to the system variable. The value is not valid for segments which have their own value assigned. With this one exception, the value remains valid, in the usual way, until a new value is assigned to the system variable. The system variables can also be assigned values by means of a function call. The same restrictions apply to these functions as to functions in the trigger. (>>> *11.11.3 "Constraints for functions in the trigger" Page 540*)
C_SPL	• With C_SPL: the end point is approximated. $APO defines the earliest point at which the approximate positioning can begin. • Without C_SPL: the motion stops exactly at the end point.

In System Software 8.2 and earlier, the identifier for approximate positioning with spline was “C_DIS”. If programs based on 8.2 or older versions are used in higher versions of 8.x and contain C_DIS, this can be retained and does not have to be changed to C_SPL.

Example

```
SPLINE
SPL P1
TRIGGER WHEN PATH=GET_PATH() ONSTART DELAY=0 DO <subprog>
PRIO=-1
SPL P2
SLIN P3
SPL P4
SCIRC P5, P6 WITH $VEL.CP=0.2
SPL P7 WITH $ACC={CP 2.0, ORI1 200, ORI2 200}
SCIRC P8, P9
SPL P10
ENDSPLINE
```

11.6.2 PTP_SPLINE ... ENDSPLINE: programming a PTP spline block

Description

PTP_SPLINE … ENDSPLINE defines a PTP spline block. A PTP spline block may contain:

- SPTP segments (number limited only by the memory capacity)
- PATH trigger
- 1 time block (TIME_BLOCK …)
- STOP WHEN PATH
- Comments
- Blank lines

The block must not include any other instructions, e.g. variable assignments or logic statements.

The start point of a spline block is the last point before the spline block. The end point of a spline block is the last point in the spline block.
A spline block does not trigger an advance run stop.

Syntax

`PTP_SPLINE` < `WITH` *SysVar1* = *Value1* <, *SysVar2* = *Value2*, … > >
Segment1
…
<*SegmentN*>
`ENDSPLINE` <`C_SPL`>

Explanation of the syntax

Element	Description
SysVar	(>>> *11.6.11 "System variables for WITH" Page 480*)
Value	Value assignment to the system variable. The value is not valid for segments which have their own value assigned. With this one exception, the value remains valid, in the usual way, until a new value is assigned to the system variable. The system variables can also be assigned values by means of a function call. The same restrictions apply to these functions as to functions in the trigger. (>>> *11.11.3 "Constraints for functions in the trigger" Page 540*)
C_SPL	• With C_SPL: the end point is approximated. $APO defines the earliest point at which the approximate positioning can begin. • Without C_SPL: the motion stops exactly at the end point.

Example

```
PTP_SPLINE WITH $ACC_AXIS[1]={CP 20, ORI1 80, ORI2 80}
SPTP P1
TRIGGER WHEN PATH=GET_PATH() ONSTART DELAY=0 DO <subprog>
PRIO=-1
SPTP P2
SPTP P3
SPTP P4 WITH $ACC_AXIS[1]={CP 10}
ENDSPLINE C_SPL
```

11.6.3 SLIN: motion programming

Description

SLIN can be programmed as a segment in a CP spline block or as an individual motion.

It is possible to copy an individual SLIN motion into a CP spline block, but only if it does not contain an assignment to system variables that are prohibited there.

Syntax

`SLIN` *End point* <`WITH` *SysVar1* = *Value1* <, *SysVar2* = *Value2*, …, >>
<`C_SPL`>

Explanation of the syntax

Element	Description
End point	Type: POS, E6POS, FRAME The coordinates refer to the BASE coordinate system. If not all components of the end point are specified, the controller takes the values of the previous position for the missing components. If this previous position is the end point of a circle with a circular angle, the values of the end point that is actually reached are applied, and not those of the programmed end point. If no previous position is known to the robot controller, the missing components are taken from the current robot position.
SysVar	(>>> *11.6.11 "System variables for WITH" Page 480*)
Value	Value assignment to the system variable. In the case of SLIN segments: The assignment applies only for this segment. The system variables can also be assigned values by means of a function call. The same restrictions apply to these functions as to functions in the trigger. (>>> *11.11.3 "Constraints for functions in the trigger" Page 540*)
C_SPL	• With C_SPL: the end point is approximated. $APO defines the earliest point at which the approximate positioning can begin. Only possible for individual motions, not for segments. • Without C_SPL: the motion stops exactly at the end point.

In System Software 8.2 and earlier, the identifier for approximate positioning with spline was “C_DIS”. If programs based on 8.2 or older versions are used in higher versions of 8.x and contain C_DIS, this can be retained and does not have to be changed to C_SPL.

11.6.4 SCIRC: motion programming

Description

SCIRC can be programmed as a segment in a CP spline block or as an individual motion.

It is possible to copy an individual SCIRC motion into a CP spline block, but only if it does not contain an assignment to system variables that are prohibited there.

Syntax

`SCIRC` *Auxiliary point*, *End point* <, `CA` *Circular angle*> <`WITH` *SysVar1* = *Value1* <,*SysVar2* = *Value2* , … >> <`C_SPL`>

Explanation of the syntax

Element	Description
Auxiliary point End point	Type: POS, E6POS, FRAME The coordinates refer to the BASE coordinate system. If not all components are specified, the controller takes the values of the previous position for the missing components. If this previous position is the end point of a circle with a circular angle, the values of the end point that is actually reached are applied, and not those of the programmed end point. The end point does not adopt values from the auxiliary point, but from the position before. If no previous position is known to the robot controller, the missing components are taken from the current robot position.
Circular angle	Unit: Degrees; without restriction (>>> *9.9 "Circular angle" Page 383*)
SysVar	(>>> *11.6.11 "System variables for WITH" Page 480*)
Value	Value assignment to the system variable. In the case of SCIRC segments: The assignment applies only for this segment. The system variables can also be assigned values by means of a function call. The same restrictions apply to these functions as to functions in the trigger. (>>> *11.11.3 "Constraints for functions in the trigger" Page 540*)
C_SPL	• With C_SPL: the end point is approximated. $APO defines the earliest point at which the approximate positioning can begin. Only possible for individual motions, not for segments. • Without C_SPL: the motion stops exactly at the end point.

In System Software 8.2 and earlier, the identifier for approximate positioning with spline was “C_DIS”. If programs based on 8.2 or older versions are used in higher versions of 8.x and contain C_DIS, this can be retained and does not have to be changed to C_SPL.

Example

```
SCIRC P2, P3 WITH $CIRC_TYPE=#PATH
```

11.6.5 SPL: motion programming

Description

SPL can be programmed as a segment in a CP spline block.

Syntax

`SPL` *End point* <`WITH` *SysVar1* = *Value1* <, *SysVar2* = *Value2* , …>>

Explanation of the syntax

Element	Description
End point	Type: POS, E6POS, FRAME The coordinates refer to the BASE coordinate system. If not all components of the end point are specified, the controller takes the values of the previous position for the missing components. If this previous position is the end point of a circle with a circular angle, the values of the end point that is actually reached are applied, and not those of the programmed end point. If no previous position is known to the robot controller, the missing components are taken from the current robot position.
SysVar	(>>> *11.6.11 "System variables for WITH" Page 480*)
Value	Value assignment to the system variable. The assignment applies only for this segment. The system variables can also be assigned values by means of a function call. The same restrictions apply to these functions as to functions in the trigger. (>>> *11.11.3 "Constraints for functions in the trigger" Page 540*)

Example

```
SPL P4 WITH $ACC={CP 2.0, ORI1 200, ORI2 200}
```

11.6.6 SPTP: motion programming

Description

SPTP can be programmed as a segment in a PTP spline block or as an individual motion.

It is possible to copy an individual SPTP motion into a PTP spline block, but only if it does not contain an assignment to system variables that are prohibited there.

Syntax

`SPTP` *End point* <`WITH` *SysVar1* = *Value1* <, *SysVar2* = *Value2* , …>>
<`C_SPL`>

Explanation of the syntax

Element	Description
End point	Type: AXIS, E6AXIS, POS, E6POS, FRAME The Cartesian coordinates refer to the BASE coordinate system. If not all components of the end point are specified, the controller takes the values of the previous position for the missing components. If this previous position is the end point of a circle with a circular angle, the values of the end point that is actually reached are applied, and not those of the programmed end point. If no previous position is known to the robot controller, the missing components are taken from the current robot position.
SysVar	(>>> *11.6.11 "System variables for WITH" Page 480*)
Value	Value assignment to the system variable. In the case of SPTP segments: The assignment applies only for this segment. The system variables can also be assigned values by means of a function call. The same restrictions apply to these functions as to functions in the trigger. (>>> *11.11.3 "Constraints for functions in the trigger" Page 540*)
C_SPL	• With C_SPL: the end point is approximated. $APO defines the earliest point at which the approximate positioning can begin. Only possible for individual motions, not for segments. • Without C_SPL: the motion stops exactly at the end point.

In System Software 8.2 and earlier, the identifier for approximate positioning with spline was “C_DIS”. If programs based on 8.2 or older versions are used in higher versions of 8.x and contain C_DIS, this can be retained and does not have to be changed to C_SPL.

11.6.7 SLIN_REL: relative motion programming

Description

SLIN_REL can be programmed as a segment in a CP spline block or as an individual motion. It is possible to copy an individual SLIN_REL motion into a CP spline block, but only if it does not contain an assignment to system variables that are prohibited there.

For information about the response of the robot controller in the case of infinitely rotating axes: (>>> *11.5.8 "REL motions for infinitely rotating axes" Page 467*)

Syntax

`SLIN_REL` *End point* <`WITH` *SysVar1* = *Value1* <, *SysVar2* = *Value2*, …, >> <`C_SPL`><`#BASE`|`#TOOL`>

Explanation of the syntax

Element	Description
End point	Type: POS, E6POS, FRAME The point must be specified in Cartesian coordinates. The controller interprets the coordinates as relative to the end point of the previous motion. If not all components of the point are specified, the controller sets the value of the missing components to 0. In other words, the absolute values of these components remain unchanged. Specifications of Status and Turn, if present, are ignored by the controller. (This is in contrast to SPTP_REL where they are taken into consideration!)
SysVar	(>>> *11.6.11 "System variables for WITH" Page 480*)
Value	Value assignment to the system variable. In the case of segments: The assignment applies only for this segment. The system variables can also be assigned values by means of a function call. The same restrictions apply to these functions as to functions in the trigger. (>>> *11.11.3 "Constraints for functions in the trigger" Page 540*)
C_SPL	• With C_SPL: the end point is approximated. $APO defines the earliest point at which the approximate positioning can begin. Only possible for individual motions, not for segments. • Without C_SPL: the motion stops exactly at the end point.
`#BASE`, `#TOOL`	• #BASE (default): The coordinates of this end point refer to the coordinate system that belongs to the physical base. • #TOOL: The coordinates of this end point refer to the coordinate system that belongs to the physical tool. $IPO_MODE has no influence on the meaning of #BASE and #TOOL.

Example

```
DECL E6POS P1 = {X 1500, Y -200, Z 2000, A 0, B 0, C 0, S
6, T27}
SPTP HOME
SLIN P1
SLIN_REL{X 0, Y 500, Z 0, A 0, B 0, C 0} WITH $BASE=$NULL-
FRAME #BASE
SLIN_REL{X 400} WITH $TOOL=$NULLFRAME C_SPL #TOOL
SLIN_REL{A 20}
SPTP_REL{A3 90} C_SPL
SPTP_REL Z 50, B -30} WITH $VEL.AXIS[4]=90 C_SPL #TOOL
SPTP_REL{A1 100}
SPLINE
    SPL P1
```

```
    SPL_REL{Z -300, B50} #TOOL
ENDSPLINE
PTPSPLINE
    SPTP P1
    SPTP_REL{A1 -100, A5 -70}
ENDSPLINE
```

11.6.8 SCIRC_REL: relative motion programming

Description

SCIRC_REL can be programmed as a segment in a CP spline block or as an individual motion. It is possible to copy an individual SCIRC_REL motion into a CP spline block, but only if it does not contain an assignment to system variables that are prohibited there.

For information about the response of the robot controller in the case of infinitely rotating axes: (>>> *11.5.8 "REL motions for infinitely rotating axes" Page 467*)

Syntax

`SCIRC_REL` *Auxiliary point*, *End point* <, `CA` *Circular angle*> <`WITH` *SysVar1* = *Value1* <,*SysVar2* = *Value2* , … >> <`C_SPL`><`#BASE`|`#TOOL`>

Explanation of the syntax

Element	Description
Auxiliary point *End point*	Type: POS, E6POS, FRAME The point must be specified in Cartesian coordinates. The controller interprets the coordinates as relative to the end point of the previous motion. If not all components of the point are specified, the controller sets the value of the missing components to 0. In other words, the absolute values of these components remain unchanged. Specifications of Status and Turn, if present, are ignored by the controller. (This is in contrast to SPTP_REL where they are taken into consideration!) At the auxiliary point, the orientation angles are also ignored. The auxiliary point cannot be approximated. The motion always stops exactly at this point.
Circular angle	Unit: Degrees; without restriction (>>> *9.9 "Circular angle" Page 383*)
SysVar	(>>> *11.6.11 "System variables for WITH" Page 480*)
Value	Value assignment to the system variable. In the case of segments: The assignment applies only for this segment. The system variables can also be assigned values by means of a function call. The same restrictions apply to these functions as to functions in the trigger. (>>> *11.11.3 "Constraints for functions in the trigger" Page 540*)

Element	Description
C_SPL	• With C_SPL: the end point is approximated. $APO defines the earliest point at which the approximate positioning can begin. Only possible for individual motions, not for segments. • Without C_SPL: the motion stops exactly at the end point.
`#BASE`, `#TOOL`	• #BASE (default): The coordinates of this end point refer to the coordinate system that belongs to the physical base. • #TOOL: The coordinates of this end point refer to the coordinate system that belongs to the physical tool. $IPO_MODE has no influence on the meaning of #BASE and #TOOL.

11.6.9 SPL_REL: relative motion programming

Description

SPL_REL can be programmed as a segment in a CP spline block. For information about the response of the robot controller in the case of infinitely rotating axes: (>>> *11.5.8 "REL motions for infinitely rotating axes" Page 467*)

Syntax

`SPL_REL` *End point* < `WITH` *SysVar1* = *Value1* <,*SysVar2* = *Value2* , … >><`#BASE`|`#TOOL`>

Explanation of the syntax

Element	Description
End point	Type: POS, E6POS, FRAME The point must be specified in Cartesian coordinates. The controller interprets the coordinates as relative to the end point of the previous motion. If not all components of the point are specified, the controller sets the value of the missing components to 0. In other words, the absolute values of these components remain unchanged. Specifications of Status and Turn, if present, are ignored by the controller. (This is in contrast to SPTP_REL where they are taken into consideration!)
SysVar	(>>> *11.6.11 "System variables for WITH" Page 480*)
Value	Value assignment to the system variable. In the case of segments: The assignment applies only for this segment. The system variables can also be assigned values by means of a function call. The same restrictions apply to these functions as to functions in the trigger. (>>> *11.11.3 "Constraints for functions in the trigger" Page 540*)

Element	Description
C_SPL	• With C_SPL: the end point is approximated. $APO defines the earliest point at which the approximate positioning can begin. Only possible for individual motions, not for segments. • Without C_SPL: the motion stops exactly at the end point.
`#BASE`, `#TOOL`	• #BASE (default): The coordinates of this end point refer to the coordinate system that belongs to the physical base. • #TOOL: The coordinates of this end point refer to the coordinate system that belongs to the physical tool. $IPO_MODE has no influence on the meaning of #BASE and #TOOL.

Example

(>>> *"Example" Page 476*)

11.6.10 SPTP_REL: relative motion programming

Description

SPTP_REL can be programmed as a segment in a PTP spline block or as an individual motion.

It is possible to copy an individual SPTP_REL motion into a PTP spline block, but only if it does not contain an assignment to system variables that are prohibited there.

For information about the response of the robot controller in the case of infinitely rotating axes: (>>> *11.5.8 "REL motions for infinitely rotating axes" Page 467*)

Syntax

`SPTP_REL` *End point* <`WITH` *SysVar1* = *Value1* <, *SysVar2* = *Value2*, ...>> <`C_SPL`><`#BASE`|`#TOOL`>

Explanation of the syntax

Element	Description
End point	Type: AXIS, E6AXIS, POS, E6POS, FRAME The end point can be specified in Cartesian or axis-specific coordinates. The controller interprets the coordinates as relative to the end point of the previous block. If not all components of the end point are specified, the controller sets the value of the missing components to 0. In other words, the absolute values of these components remain unchanged. Specifications of Status and Turn, if present, are taken into consideration by the controller. (This is in contrast to SLIN_REL, SCIRC_REL and SPL_REL where they are ignored!)
SysVar	(>>> *11.6.11 "System variables for WITH" Page 480*)

Element	Description
Value	Value assignment to the system variable. In the case of SPTP segments: The assignment applies only for this segment. The system variables can also be assigned values by means of a function call. The same restrictions apply to these functions as to functions in the trigger. (>>> *11.11.3 "Constraints for functions in the trigger" Page 540*)
C_SPL	• With C_SPL: the end point is approximated. $APO defines the earliest point at which the approximate positioning can begin. Only possible for individual motions, not for segments. • Without C_SPL: the motion stops exactly at the end point.
`#BASE`, `#TOOL`	Only permissible if the end point was specified in Cartesian coordinates. • #BASE (default): The coordinates of this end point refer to the coordinate system that belongs to the physical base. • #TOOL: The coordinates of this end point refer to the coordinate system that belongs to the physical tool. $IPO_MODE has no influence on the meaning of #BASE and #TOOL.

Example

(>>> *"Example" Page 476*)

11.6.11 System variables for WITH

Spline block, individual spline motion

For spline blocks and individual spline motions, it is possible to write to the following system variables using the WITH line:

$ACC
$ACC_AXIS
$ACC_EXTAX
$APO
$BASE
$CIRC_TYPE
$COLLMON_TOL_PRO
$DERI
$ECO_LEVEL
$EXT_IPO_MODE
$GEAR_JERK
$IPO_MODE
$JERK
$JOINT_OFFSET
$LASER_MODE

$LOAD
$LOAD_A1, $LOAD_A2, $LOAD_A3
$ORI_TYPE
$ROTSYS
$SPL_ORI_JOINT_AUTO
$SPL_VEL_MODE
$SPL_VEL_RESTR
$SYNC_ID
$SYNC_LIST
$TENSION
$TOOL
$VEL
$VEL_AXIS
$VEL_EXTAX
Additionally for SCIRC: $CIRC_MODE

Spline segment

For spline segments, it is possible to write to the following system variables using the WITH line:
$ACC
$ACC_AXIS
$ACC_EXTAX
$CIRC_TYPE
$COLLMON_TOL_PRO
$DERI
$EX_AX_IGNORE
$GEAR_JERK
$JERK
$ORI_TYPE
$ROTSYS
$SYNC_ID
$TENSION
$VEL
$VEL_AXIS
$VEL_EXTAX
Additionally for SCIRC: $CIRC_MODE

11.6.12 TIME_BLOCK: programming a time block for spline

Description

TIME_BLOCK can be used in CP and PTP spline blocks.

TIME_BLOCK can be used to execute the spline block or part of one in a defined time. It is also possible to allocate time components to areas of TIME_BLOCK.

Points can be modified in, added to or removed from the spline block without changing the time specifications. This enables the user to correct the Cartesian path and retain the existing timing.

A spline block may include 1 time block, i.e. 1 statement of the type TIME_BLOCK START … TIME_BLOCK END. This, in turn, may contain

any number of TIME_BLOCK PART statements. The time block may only be used in spline blocks.

A CP spline block can contain either 1 time block or 1 constant velocity range, but not both.

Syntax

```
SPLINE
<Spline segments…>
…
TIME_BLOCK START
Spline segment
<Spline segments…>
...
< <TIME_BLOCK PART = Component_1>
…
Spline segment
<Spline segments…>
...
TIME_BLOCK PART = Component_N>
TIME_BLOCK END = Overall time
<Spline segments…>
…
ENDSPLINE
```

Explanation of the syntax

It is not essential for there to be spline segments before TIME_BLOCK START and after TIME_BLOCK END. It is nonetheless advisable to program as follows:

- There is at least 1 spline segment between SPLINE and TIME_BLOCK START.
- There is at least 1 spline segment between TIME_BLOCK END and ENDSPLINE.

Advantages:

- The programmed overall time is maintained exactly even in the case of approximate positioning.
- Segments before TIME_BLOCK START make it possible to accelerate to the required velocity.

Element	Description
Component	Type: INT or REAL; constant, variable or function Desired component of *Overall time* for the following distance: • From the point before `TIME_BLOCK PART=`*Previous_component* to the point before `TIME_BLOCK PART=`*Component* • If *Previous_component* does not exist: From the point before `TIME_BLOCK START` to the point before `TIME_BLOCK PART=`*Component* "Desired component" means: the components are maintained as accurately as possible by the robot controller. Generally, however, they are not maintained exactly. The user can assign the components in such a way that they add up to 100. The components can then be considered as percentages of *Overall time*. The components do not have to add up to 100, however, and can have any sum! The robot controller always equates the sum of the components to *Overall time*. This allows the components to be used very flexibly and also changed. If components are assigned, there must always be a `TIME_BLOCK PART` directly before `TIME_BLOCK END`. There must be no segments in between.
Overall time	Type: INT or REAL; constant, variable or function; unit: s Time in which the following distance is traveled: • From the point before `TIME_BLOCK START` to the point before `TIME_BLOCK END` The value must be greater than 0. The overall time is maintained exactly. If this time cannot be maintained, e.g. because too short a time has been programmed, the robot executes the motion in the fastest possible time. In T1 and T2, a message is also displayed.

If the value for *Component* or *Overall time* is assigned via a function, the same restrictions apply as for the functions in the trigger.
(>>> *11.11.3 "Constraints for functions in the trigger" Page 540*)

Example

```
SPLINE
SLIN P1
SPL P2
TIME_BLOCK START
SLIN P3
TIME_BLOCK PART = 12.7
SPL P4
SPL P5
SPL P6
TIME_BLOCK PART = 56.4
SCIRC P7, P8
SPL P9
TIME_BLOCK PART = 27.8
```

```
TIME_BLOCK END = 3.9
SLIN P10
ENDSPLINE
```

Points P2 to P9 are executed exactly in the programmed time of 3.9 s. The robot controller equates the overall time of 3.9 s to the sum of all components, i.e. 96.9.

Distance	Time assigned by the robot controller to the distance
P2 … P3	12.7 components of 3.9 s = 0.51 s
P3 … P6	56.4 components of 3.9 s = 2.27 s
P6 … P9	27.8 components of 3.9 s = 1.12 s

Block selection

Whether or not the robot controller plans the time block depends on the line to which a block selection is carried out.

Block selection to the line ...	Time block is planned?
in the spline block before TIME_BLOCK START	Yes
TIME_BLOCK START	No The spline block is executed as if there were no TIME_BLOCK statements present.
in the time block	
TIME_BLOCK END	
in the spline block after TIME_BLOCK END	

If the robot controller does not plan the time block, it generates the following message: *Time block ignored due to BCO run.*

$PATHTIME

The data of the time-based spline can be read via the system variable $PATHTIME. $PATHTIME is filled with the data as soon as the robot controller has completed the planning of the spline block. The data are retained until the next spline block has been planned.

$PATHTIME is a structure and consists of the following components:

Component	Description
REAL $PATHTIME.TOTAL	Time actually required for the entire spline block (s)
REAL $PATHTIME.SCHEDULED	Overall time planned for the time block (s)
REAL $PATHTIME.PROGRAMMED	Overall time programmed for the time block (s)
INT $PATHTIME.N_SECTIONS	Number N of TIME_BLOCK_PART lines
REAL $PATHTIME.MAX_DEV	Maximum deviation of all TIME_BLOCK_PARTs between the programmed time and the planned time (%)
INT $PATHTIME.MAX_DEV_SECTION	Number of the TIME_BLOCK_PART with the greatest deviation between the programmed time and the planned time

11.6.13 CONST_VEL: programming a constant velocity range for spline motions

Description

CONST_VEL is used to define constant velocity ranges. CONST_VEL can only be used in CP spline blocks.

There must be at least 1 spline segment between CONST_VEL END and ENDSPLINE.

A CP spline block can contain either 1 CONST_VEL or 1 TIME_BLOCK, but not both.

> Basic information about the constant velocity range can be found here: (>>> *10.4.6 "Constant velocity range in the CP spline block" Page 423*)

Syntax

`CONST_VEL START =` *Offset* `<ONSTART>`

<Spline segments…>

...

`CONST_VEL END =` *Offset* `<ONSTART>`

Explanation of the syntax

Element	Description
`ONSTART`	Reference point of the statement • With ONSTART: Start point • Without ONSTART: End point If the start or end point is approximated, the reference point is generated in the same way as for homogenous approximate positioning with the PATH trigger. (>>> *11.11.2.2 "Reference point for homogenous approximate positioning" Page 537*)
Offset	Type: INT or REAL; constant, variable or function; unit: mm The start of the range can be shifted in space by means of CONST_VEL START = *Offset*. The end of the range can be shifted in space by means of CONST_VEL END = *Offset*. • Positive value: Offset towards the end of the motion • Negative value: Offset towards the start of the motion The point to which the offset refers depends on whether ONSTART is set or not. If no offset is desired, *Offset*=0 must be programmed. (>>> *10.4.6.2 "Maximum limits" Page 423*)

> The value for *Offset* can be assigned using a function. The same restrictions apply as for the functions in the trigger.
> (>>> *11.11.3 "Constraints for functions in the trigger" Page 540*)

Example

Here, the constant velocity range extends over several segments with different programmed velocities. In this case, the lowest of the velocities, i.e. 0.2 m/s, is valid for the whole range.

```
1 PTP P0
2 SPLINE WITH $VEL.CP = 2.5
3 SLIN P1
4 CONST_VEL START = +100
5 SPL P2 WITH $VEL.CP = 0.5
6 SLIN P3 WITH $VEL.CP = 0.2
7 SPL P4 WITH $VEL.CP = 0.4
8 CONST_VEL END = -50
9 SCIRC P5, P6
10 SLIN P7
11 ENDSPLINE
```

11.6.13.1 System variables for CONST_VEL

$STOP_CONST_VEL_RED

If the maximum possible constant velocity in a constant velocity range is below the programmed velocity, the robot controller generates one of the following messages:

- *In the CONST_VEL END range, instead of $VEL.CP={Setpoint $VEL.CP} m/s only {Velocity reached} m/s because line {Line of the limiting segment} reached.*
- *In CONST_VEL START instead of $VEL.CP={Setpoint $VEL.CP} m/s only {Velocity reached} m/s because line {Line of the limiting segment} reached.*
- *In CONST_VEL END instead of $VEL.CP={Setpoint $VEL.CP} m/s only {Velocity reached} m/s because line {Line of the limiting segment} reached.*

For operating modes T1/T2, it is possible to configure whether these are notification messages or acknowledgement messages. This is carried out using the system variable $STOP_CONST_VEL_RED.

Value	Description
FALSE (default)	Notification message
TRUE	• Operating mode T1 or T2: Acknowledgement message The robot stops. In the program, the block pointer indicates the spline segment that triggered the stop. Program execution cannot be resumed until the operator has acknowledged the message. • Operating mode AUT or AUT EXT (KSS) or EXT (VSS): Notification message

$STOP_CONST_VEL_RED is initialized with FALSE in the case of a cold start or program selection, but not if a program is reset.

$STOP_CONST_VEL_RED can be modified via the variable correction function or KRL.

$CONST_VEL

$CONST_VEL specifies the velocity (mm/s) in the constant velocity range for the CP spline block that is currently being planned. The value remains valid until a different spline block with constant velocity range is planned.

$CONST_VEL is write-protected. $CONST_VEL is invalid if one of the following motions is currently being planned:

- CP spline block without constant velocity range
- PTP spline block
- Individual spline motion
- PTP, LIN, CIRC

$CONST_VEL_C

$CONST_VEL_C corresponds to $CONST_VEL, but with the difference that $CONST_VEL_C refers to the CP spline block that is currently being executed.

Possible practical use:

The value of $CONST_VEL can be written to a user-specific local variable for each constant velocity range. This provides an overview of the constant velocities that can be reached. The application can then be programmed accordingly, e.g. the point in time at which a nozzle is to be opened, or similar.

11.6.14 STOP WHEN PATH: programming a conditional stop for spline

Description

The operator can program a conditional stop with STOP WHEN PATH.

Further information about the conditional stop can be found in this documentation.
(>>> *10.4.5 "Conditional stop" Page 419*)

Positions

The conditional stop can be used in the following positions:

- In the individual spline block
- In the spline block (CP and PTP)

  There must be at least 1 segment between STOP WHEN PATH and ENDSPLINE.
- Before a spline block (CP and PTP)

  STOP WHEN PATH refers to the spline block in this case. There may be statements between STOP WHEN PATH and the spline block, but no motion instructions.

Syntax

`STOP WHEN PATH =` *Offset* `<ONSTART> IF` *Condition*

Explanation of the syntax

Element	Description
`ONSTART`	Reference point of the statement • Without ONSTART: End point • With ONSTART: Start point If the reference point is approximated, the same rules apply as for the PATH trigger. (>>> *11.11.2.1 "Reference point for approximate positioning – overview" Page 537*)
Offset	Type: INT or REAL; constant, variable or function; unit: mm The stop point can be shifted in space by means of *Offset*. • Positive value: Shift towards the end of the motion • Negative value: Shift towards the start of the motion The point to which the offset refers depends on whether ONSTART is set or not. If no offset is desired, *Offset*=0 must be programmed. There are limits to the distance the stop point can be offset. The same limits apply as for the PATH trigger. (>>> *"Max. offset" Page 534*)
Condition	Type: BOOL Stop condition. The following are permitted: • a global Boolean variable • a signal name • a comparison • a simple logic operation: NOT, OR, AND or EXOR

The value for *Offset* can be assigned using a function. The same restrictions apply as for the functions in the trigger.
(>>> *11.11.3 "Constraints for functions in the trigger" Page 540*)

11.6.15 $EX_AX_IGNORE

Description

$EX_AX_IGNORE can only be used in the WITH line of spline segments.

Each bit of $EX_AX_IGNORE corresponds to an external axis number. If a specific bit is set to the value “1”, the robot controller ignores the taught or programmed position of this external axis at the end point of the segment. Instead, the robot controller calculates the optimal position for this point on the basis of the surrounding external axis positions.

Recommendation: Whenever no specific position of the external axis is required for a point, use $EX_AX_IGNORE and set the bit for that external axis to the value “1”. This reduces the cycle time.

In the program run modes MSTEP and ISTEP, the robot stops at the positions calculated by the robot controller.

In the case of a block selection to a point with “$EX_AX_IGNORE = Bit n = 1”, the robot adopts the position calculated by the robot controller.

"$EX_AX_IGNORE = Bit n = 1" is not allowed for the following segments:

- For the first segment in a spline block (only up to KUKA System Software 8.2)
- For the last segment in a spline block
- In the case of successive segments with identical Cartesian end points, "$EX_AX_IGNORE = Bit n = 1" is not allowed for the first and last segments. (only up to and including KUKA System Software 8.2)

From KUKA System Software 8.3 onwards: If $EX_AX_IGNORE is programmed for an SPTP segment and the external axis concerned is mathematically coupled, the robot controller rejects $EX_AX_IGNORE. The taught or programmed position of that axis is taken into consideration. In T1/T2, the robot controller generates the following message: *Reject $EX_AX_IGNORE in line {Block number} because {External axis number} is mathematically coupled.*

Syntax

`$EX_AX_IGNORE=`*Bit array*

Explanation of the syntax

Element	Description
Bit array	• **Bit n = 1**: Taught/programmed position of the external axis is ignored. • **Bit n = 0**: Taught/programmed position of the external axis is taken into consideration.

Bit n	5	4	3	2	1	0
Axis	E6	E5	E4	E3	E2	E1

Example

```
SPLINE
SPL P1
SPL P2
SLIN P3 WITH $EX_AX_IGNORE = 'B000001'
SPL P4
ENDSPLINE
```

For P3, the robot controller ignores the taught position of external axis E1.

11.7 Program execution control

11.7.1 CONTINUE: preventing an advance run stop

Description

CONTINUE can be used to prevent an advance run stop that would otherwise occur in the following program line.

CONTINUE always applies to the following line, even if this is a blank line! Exception: If the following line contains the statement ON_ERROR_PROCEED, CONTINUE applies to the line after.

Syntax

```
CONTINUE
```

Examples

Preventing both advance run stops:

```
CONTINUE
$OUT[1]=TRUE
CONTINUE
$OUT[2]=FALSE
```

In this case, the outputs are set in the advance run. When exactly they are set cannot be foreseen.

ON_ERROR_PROCEED with CONTINUE:

```
ON_ERROR_PROCEED
CONTINUE
$OUT[1]=TRUE
```

```
CONTINUE
ON_ERROR_PROCEED
$OUT[1]=TRUE
```

The effect of both sequences of statements is identical. In both examples, ON_ERROR_PROCEED and CONTINUE act on $OUT[1]=TRUE.

11.7.2 EXIT: exiting a loop

Description

Exit from a loop. The program is then continued after the loop. EXIT may be used in any loop.

Syntax

```
EXIT
```

Example

The loop is exited when `$IN[1]` is set to TRUE. The program is then continued after ENDLOOP.

```
DEF EXIT_PROG()
PTP HOME
LOOP
    PTP POS_1
    PTP POS_2
    IF $IN[1] == TRUE THEN
        EXIT
    ENDIF
    CIRC HELP_1, POS_3
    PTP POS_4
ENDLOOP
PTP HOME
END
```

11.7.3 FOR ... TO ... ENDFOR: programming a counting loop

Description

A statement block is repeated until a counter exceeds or falls below a defined value.

After the last execution of the statement block, the program is resumed with the first statement after ENDFOR. The loop execution can be exited prematurely with EXIT.

Loops can be nested. In the case of nested loops, the outer loop is executed completely first. The inner loop is then executed completely.

Syntax

`FOR` *Counter* = *Start value* `TO` *End value* `<STEP` *Increment*`>`

`<`*Statements*`>`

`ENDFOR`

Explanation of the syntax

Element	Description
Counter	Type: INT Variable that counts the number of times the loop has been executed. The preset value is *Start*. The variable must first be declared. The value of *Counter* can be used in statements inside and outside of the loop. Once the loop has been exited, *Counter* retains its most recent value.
Start; End	Type: INT *Counter* must be preset to the value *Start*. Each time the loop is executed, the value of *Counter* is automatically increased by the increment. If the value exceeds or falls below the *End* value, the loop is terminated.
Increment	Type: INT Value by which *Counter* is changed every time the loop is executed The value may be negative. Default value: 1. • Positive value: the loop is ended if *Counter* is greater than *End*. • Negative value: the loop is ended if *Counter* is less than *End*. The value may not be either zero or a variable.

Example

The variable `B` is incremented by 1 after each of 5 times the loop is executed.

```
INT A
...
FOR A=1 TO 10 STEP 2
    B=B+1
ENDFOR
```

11.7.4 GOTO: jump to position in the program

Description

Unconditional jump to a specified position in the program. Program execution is resumed at this position.

The destination must be in the same subprogram or function as the GOTO statement.

The following jumps are not possible:

- Into an IF statement from outside.
- Into a loop from outside.
- From one CASE statement to another CASE statement.

GOTO statements lead to a loss of structural clarity within a program. It is better to work with IF, SWITCH or a loop instead.

Syntax

`GOTO` *Label*

`...`

Label`:`

Explanation of the syntax

Element	Description
Label	Position to which a jump is made. At the destination position, *Label* must be followed by a colon.

Example 1

Unconditional jump to the program position `GLUESTOP`.

```
GOTO GLUESTOP
...
GLUESTOP:
```

Example 2

Unconditional jump from an IF statement to the program position `END`.

```
IF X>100 THEN
   GOTO ENDE
ELSE
   X=X+1
ENDIF
A=A*X
...
ENDE:
END
```

11.7.5 HALT: stopping a program

Description

Stops the program. The last motion instruction to be executed will, however, be completed.

Execution of the program can only be resumed using the Start key. The next instruction after HALT is then executed.

In an interrupt program, program execution is only stopped after the advance run has been completely executed.

Syntax

`HALT`

11.7.6 IF ... THEN ... ENDIF: programming a conditional branch

Description

Conditional branch. Depending on a condition, either the first statement block (THEN block) or the second statement block (ELSE block) is executed. The program is then continued after ENDIF.

The ELSE block may be omitted. If the condition is not satisfied, the program is then continued at the position immediately after ENDIF.

There is no limit on the number of statements contained in the statement blocks. Several IF statements can be nested in each other.

Syntax

IF *Condition* THEN
Statements
<ELSE
Statements>
ENDIF

Explanation of the syntax

Element	Description
Condition	Type: BOOL Possible: • Variable of type BOOL • Function of type BOOL • Logic operation, e.g. a comparison, with a result of type BOOL

Example 1

IF statement without ELSE

```
IF A==17 THEN
    B=1
ENDIF
```

Example 2

IF statement with ELSE

```
IF $IN[1]==TRUE THEN
    $OUT[17]=TRUE
ELSE
    $OUT[17]=FALSE
ENDIF
```

11.7.7 LOOP ... ENDLOOP: programming an endless loop

Description

Loop that endlessly repeats a statement block. The loop execution can be exited with EXIT.

Loops can be nested. In the case of nested loops, the outer loop is executed completely first. The inner loop is then executed completely.

Syntax

```
LOOP
Statements
ENDLOOP
```

Example

The loop is executed until input $IN[30] is set to true.

```
LOOP
    LIN P_1
    LIN P_2
    IF $IN[30]==TRUE THEN
        EXIT
    ENDIF
ENDLOOP
```

11.7.8 ON_ERROR_PROCEED: Skipping an instruction that causes an interpreter error

Description

ON_ERROR_PROCEED leads to a situation where the following program line is skipped if it causes an interpreter error. Consequence:

- The statement that causes the error is not executed.
- Although the controller plans the associated error message, it does not actually issue it.
- The responses associated with the message (e.g. robot stops) do not occur.

ON_ERROR_PROCEED always applies to the following line, even if this is a blank line! Exception: If the following line contains the statement CONTINUE, ON_ERROR_PROCEED applies to the line after.

If the line after ON_ERROR_PROCEED is a subprogram call, the statement then refers to the call itself, and not to the first line of the subprogram.

ON_ERROR_PROCEED can be used in the robot interpreter, including interrupt programs.

WARNING

The program sequence can vary depending on whether an error to which ON_ERROR_PROCEED reacts occurs or not. This is particularly true if it is a motion block that is skipped due to ON_ERROR_PROCEED. This changes the entire motion sequence.
The layout of the system must be matched to the potential sequence variants. The potential variants must be induced and tested during the program tests. Failure to observe this may result in death to persons, injuries or damage to property.

Limitations

ON_ERROR_PROCEED has no impact on the following instructions.

Motions:

- For spline motions
- MOVE_EMI

Control structures:

- FOR / ENDFOR
- GOTO
- IF /ELSE / ENDIF
- LOOP / ENDLOOP
- REPEAT / UNTIL
- SKIP / ENDSKIP
- SWITCH / CASE / DEFAULT / ENDSWITCH
- WHILE / ENDWHILE

Other:

- ASYPTP; ASYSTOP; ASYCONT; ASYCANCEL

$ERR, ERR_CLEAR(), ERR_RAISE()

Important aids for working with ON_ERROR_PROCEED:

- **$ERR**

  $ERR allows the user to see whether planning has detected an error (even if the message has been suppressed due to ON_ERROR_PROCEED) and if so, what error it is. If necessary, the user can program their own error strategy in KRL.

  (>>> *11.7.8.4 "$ERR" Page 498*)
- **ERR_CLEAR()**

  Deletes the contents of $ERR.
- **ERR_RAISE()**

  Subsequently output the suppressed message after all

ERR_CLEAR() and ERR_RAISE() can only process $ERR or a variable derived from $ERR as an OUT parameter.

Example with CONTINUE

ON_ERROR_PROCEED with CONTINUE:

```
ON_ERROR_PROCEED
CONTINUE
$OUT[1]=TRUE
```

```
CONTINUE
ON_ERROR_PROCEED
$OUT[1]=TRUE
```

The effect of both sequences of statements is identical. In both examples, ON_ERROR_PROCEED and CONTINUE act on $OUT[1]=TRUE.

Further examples

(>>> *11.7.8.3 "Examples for ON_ERROR_PROCEED" Page 497*)

11.7.8.1 Handlable/non-handlable errors

Handlable errors

Errors that are detected during motion planning can be handled with ON_ERROR_PROCEED.

Examples:

- *Static gear torque overload {Axis number} by {% value}* (PTP)

- *Software limit switch point cannot be reached {Motion direction}{Axis number}*
- *Auxiliary point identical to end point* (CIRC)

Errors that are thrown during motion planning can also be handled without them being directly related to motion planning. The reason for this is that they occur in the interpreter.

Example:

- *Trigger switching action inadmissible for this motion type*

Non-handlable errors

Interpolation errors or errors relating to other monitoring tasks that are only thrown when the motion is executed cannot be handled using ON_ERROR_PROCEED.

Examples:

- *Command gear torque {Axis number}*
- *STOP due to software limit switch {Motion direction} {Axis number}*
- *Ackn.: Collision detection axis {Axis number}*

11.7.8.2 ON_ERROR_PROCEED with motion instructions

The following motion instructions can be handled using ON_ERROR_PROCEED:

- LIN, LIN_REL
- PTP, PTP_REL
- CIRC, CIRC_REL

Motion parameters

Triggers:

Triggers are interpreted as being permanently associated with a motion block. If a block is not executed due to ON_ERROR_PROCEED, then its triggers are discarded.

Other motion parameters:

All other motion parameters such as tool, base, velocities, acceleration, load, filters, etc. are not specially handled.

This means that if parameters such as these are programmed before a block that is not executed due to ON_ERROR_PROCEED, then these parameters remain in place and are executed along with the next block.

Example:

```
$TOOL=TOOL_DATA[5]
ON_ERROR_PROCEED
LIN P1
LIN P2
```

If the line LIN P1 does not cause an interpreter error, then it is executed. $TOOL=TOOL_DATA[5] comes into effect for P1.

If the line LIN P1 causes an interpreter error, then it is not executed. $TOOL=TOOL_DATA[5] then comes into effect for P2.

Approximation

Skipping a motion block by way of ON_ERROR_PROCEED does not interfere with the programmed approximation of the preceding block.

This means that if a block has registered approximation (C_PTP, C_DIS, C_VEL, C_ORI) and the subsequent block is skipped due to ON_ERROR_PROCEED, then an attempt is made to blend into the following block.

Example:

```
LIN P1 C_DIS
ON_ERROR_PROCEED
LIN P2
LIN P3
```

If the line LIN P2 does not cause an interpreter error, then it is executed. Approximate positioning from P1 to P2.

If the line LIN P2 causes an interpreter error, then it is not executed. Approximate positioning from P1 to P3.

11.7.8.3 Examples for ON_ERROR_PROCEED

Example

If you do not wish to suppress all interpreter error messages, but only specific ones, this distinction can be made using $ERR.NUMBER.

In this example, only message 1422 is suppressed. Other interpreter error messages are displayed.

```
1  DEF myProg ()
2  DECL E6POS myPos
3  INI
4  ON_ERROR_PROCEED
5  myPos = $POS_INT
6  SWITCH($ERR.NUMBER)
7    CASE 0
8    CASE 1422
9        ;program fault service function if required
10    DEFAULT
11        ERR_RAISE($ERR)
12 ENDSWITCH
    ...
13 END
```

Line	Description
4, 5	Line 5 actually triggers the message 1422 *{$variable} value invalid*. (Unless the program is called by an interrupt.) ON_ERROR_PROCEED in the preceding line suppresses the message.
6 … 12	Differentiation dependent on $ERR.NUMBER
7	If no error occurred in line 5, $ERR.NUMBER==0. In this case, no action is required.
8, 9	If message 1422 has been triggered, $ERR.NUMBER==1422. If required, a fault service function can be programmed.
10, 11	If a message other than 1422 was triggered, this message is now (subsequently) generated via ERR_RAISE.

Example 2

This example shows that each program level has its own representation of $ERR. The example also shows how the $ERR information can be transferred to a different level.

```
1 DEF myMainProg2 ()
2 INI
3 ON_ERROR_PROCEED
4 $OUT[-10] = TRUE
5 myHandleErr ($ERR, $ERR)
6 END
---------------------------------------
7 DEF myHandleErr (inErr:IN, outErr:OUT)
8 DECL Error_T inErr, outErr
9 ON_ERROR_PROCEED
10 $OV_PRO=100/0
11 ERR_RAISE($ERR)
12 ERR_RAISE(outErr)
13 ERR_RAISE(inErr)
...
14 END
```

Line	Description
3, 4	Line 4 actually triggers the message 1444 *Array index inadmissible*. ON_ERROR_PROCEED in the preceding line suppresses the message.
5, 7	The contents of $ERR are transferred to a subprogram twice: once as an IN parameter and once as an OUT parameter.
9, 10	Line 10 actually triggers the message 1451 *Division by 0*. ON_ERROR_PROCEED in the preceding line suppresses the message.
11	ERR_RAISE($ERR) generates the message from line 10, and not that from line 4. $ERR always has the information from the current level. In this case, from the subprogram myHandleErr.
12	ERR_RAISE(outErr) generates the message from line 4 of the main program, as outErr is a reference to $ERR in the main program.
13	ERR_RAISE(inErr) is not permissible and thus triggers the message 1451 *{(Variable name)} invalid argument*. ERR_RAISE can only process $ERR or an OUT variable derived from $ERR.

11.7.8.4 $ERR

Description

Structure with information about the current program

The variable can be used to evaluate the currently executed program relative to the advance run. For example, the variable can be used to evaluate errors in the program in order to be able to respond to them with a suitable error strategy.

The variable is write-protected and can only be read.

$ERR exists separately for the robot and submit interpreters. Each interpreter can only access its own variable. $ERR does not exist for the command interpreter.

Each subprogram level has its own representation of $ERR. In this way, the information from one level does not overwrite the information from different levels and information can be read from different levels simultaneously.

ON_ERROR_PROCEED implicitly deletes the information from $ERR in the current interpreter and at the current level.

Syntax

Information=`$ERR`

Explanation of the syntax

Element	Description
Information	Type: Error_T List with information about the program currently being executed

Error_T

```
STRUC Error_T INT number, PROG_INT_E interpreter,
INT_TYP_E int_type, INT int_prio, line_nr, CHAR mod-
ule[24], up_name[24], TRIGGER_UP_TYPE trigger_type
```

Element	Description
Number	Only in the event of an interpreter error: Message number If no error has occurred, the value zero is displayed.
Interpreter	Current interpreter • #R_INT: Robot interpreter • #S_INT: System submit interpreter • #EXT_S_INT1: extended submit interpreter 1 • #EXT_S_INT2: extended submit interpreter 2 • …
int_type	Current program type and interrupt state • #I_NORMAL: The program is not an interrupt program. • #I_INTERRUPT: The program is an interrupt program. • #I_STOP_INTERRUPT: Interrupt by means of $STOPMESS (error stop)
int_prio	Priority of the interrupt
line_nr	Only in the event of an interpreter error: Number of the line that triggered the error **Note**: The number does not generally correspond to the line number in the smartHMI program editor! In order to understand the numbering, open the program with a simple editor and do not count lines that start with “&”. If no error has occurred, the value zero is displayed.
module[]	Name of the current program

Element	Description
up_name[]	Name of the current subprogram
trigger_type	Context in which the trigger belonging to a subprogram was triggered • #TRG_NONE: The subprogram is not a trigger subprogram. • #TRG_REGULAR: The trigger subprogram was switched during forward motion. • #TRG_BACKWARD: The trigger subprogram was switched during backward motion. • #TRG_RESTART: The trigger subprogram was switched on switching back to forward motion. • #TRG_REPLAY: The trigger subprogram was switched repeatedly after backward motion.

Example

$ERR can be used not only for error treatment, but also to determine the current surroundings.

In this example, a parameter is transferred to a subprogram from both a robot program and a submit program. In the subprogram, the system determines which interpreter the parameter came from. The action that is carried out depends on the result.

Robot program:

```
DEF Main ()
...
my_prog (55)
...
END
```

Submit program:

```
DEF my_sub ()
...
LOOP
     my_prog (33)
     ...
ENDLOOP
...
END
```

Subprogram:

```
GLOBAL DEF my_prog (par:IN)
INT par
INI
SWITCH $ERR.interpreter
     CASE #R_INT
        $OUT[par] = TRUE
     CASE #S_INT
        sub_prog_s()
     CASE #EXT_S_INT1
        sub_prog_1()
     CASE #EXT_S_INT2
        sub_prog_2()
     CASE #EXT_S_INT3
        sub_prog_3()
```

```
      ...
ENDSWITCH
...
END
```

Example 2

This example illustrates that each program level has its own representation of $ERR.

```
1 DEF myMainProg ()
2 INT myVar, myVar2
3 INI
4 ON_ERROR_PROCEED
5 mySubProg (myVar)
6 HALT
7 myVar2 = 7
8 mySubProg (myVar2)
9 END
---------------------------------------
10 DEF mySubProg (myTest:IN)
11 INT myTest
12 HALT
13 END
```

Line	Description
4, 5	Line 5 actually triggers the message 1422 *{$variable} value invalid* because myVar is not initialized and can thus not be transferred to a subprogram. ON_ERROR_PROCEED in the preceding line suppresses the message.
6	If $ERR is read here using the variable correction function, the following components have the following values: $ERR.number == 1422 $ERR.line_nr == 15 $ERR.module[] == "MYMAINPROG" $ERR.up_name[] == "MYMAINPROG"
12	If $ERR is read here in the subprogram using the variable correction function, the following components have the following values: $ERR.number == 0 $ERR.line_nr == 0 $ERR.module[] == "MYMAINPROG" $ERR.up_name[] == "MYSUBPROG" This makes it clear that: $ERR always has the information from the current level (in this case, from the subprogram MySubProg). The information from MyMainProg, on the other hand, is unknown.

11.7.9 REPEAT ... UNTIL: programming a post-test loop

Description

Post-test (non-rejecting) loop. Loop that is repeated until a certain condition is fulfilled.

The statement block is executed at least once. The condition is checked after each loop execution. If the condition is met, program execution is resumed at the first statement after the UNTIL line.

Loops can be nested. In the case of nested loops, the outer loop is executed completely first. The inner loop is then executed completely.

Syntax

`REPEAT`

Statements

`UNTIL` *Termination condition*

Explanation of the syntax

Element	Description
Break condition	Type: BOOL Possible: • Variable of type BOOL • Function of type BOOL • Logic operation, e.g. a comparison, with a result of type BOOL

Example 1

The loop is to be executed until $IN[1] is true.

```
R=1
REPEAT
    R=R+1
UNTIL $IN[1]==TRUE
```

Example 2

The loop is executed once, even though the termination condition is already fulfilled before the loop execution, because the termination condition is not checked until the end of the loop. After execution of the loop, `R` has the value 102.

```
R=101
REPEAT
    R=R+1
UNTIL R>100
```

11.7.10 SWITCH ... CASE ... ENDSWITCH: programming a multiple branch

Description

Selects one of several possible statement blocks, according to a selection criterion. Every statement block has at least one identifier. The block whose identifier matches the selection criterion is selected.

Once the block has been executed, the program is resumed after ENDSWITCH.

If no identifier agrees with the selection criterion, the DEFAULT block is executed. If there is no DEFAULT block, no block is executed and the program is resumed after ENDSWITCH.

The SWITCH statement cannot be prematurely exited using EXIT.

Syntax

`SWITCH` *Selection criterion*

`CASE` *Identifier1* `<,`*Identifier2*`,...>`

Statement block

`<CASE` *IdentifierM* `<,`*IdentifierN*`,...>`

Statement block `>`

`<DEFAULT`

Default statement block`>`

`ENDSWITCH`

DEFAULT may only occur once in a SWITCH statement.

Explanation of the syntax

Element	Description
Selection criterion	Type: INT, CHAR, ENUM This can be a variable, a function call or an expression of the specified data type.
Identifier	Type: INT, CHAR, ENUM The data type of the identifier must match the data type of the selection criterion. A statement block can have any number of identifiers. Multiple block identifiers must be separated from each other by a comma.

Example 1

Selection criterion and identifier are of type INT.

```
INT VERSION
...
SWITCH VERSION
    CASE 1
        UP_1()
    CASE 2,3
        UP_2()
        UP_3()
        UP_3A()
    DEFAULT
        ERROR_UP()
ENDSWITCH
```

Example 2

Selection criterion and identifier are of type CHAR. The statement `SP_5()` is never executed here because the identifier `C` has already been used.

```
SWITCH NAME
    CASE "A"
        UP_1()
```

```
    CASE "B","C"
        UP_2()
        UP_3()
    CASE "C"
        UP_5()
ENDSWITCH
```

11.7.11 WAIT FOR … : waiting until a condition has been met

Description

WAIT FOR stops the program until a specific condition is fulfilled. Program execution is then resumed.

WAIT FOR triggers an advance run stop.

> If, due to incorrect formulation, the expression can never take the value TRUE, the compiler does not recognize this. In this case, execution of the program will be permanently halted because the program is waiting for a condition that cannot be fulfilled.

Syntax

`WAIT FOR` *Condition*

Explanation of the syntax

Element	Description
Condition	Type: BOOL Condition, the fulfillment of which allows program execution to be resumed. • If the condition is FALSE, program execution is stopped until the condition is TRUE. • If the condition is already TRUE when WAIT is called, program execution is not halted.

Examples

Interruption of program execution until `$IN[17]` is TRUE:

```
WAIT FOR $IN[17]
```

Interruption of program execution until `BIT1` is FALSE:

```
WAIT FOR BIT1==FALSE
```

11.7.12 WAIT SEC … : programming a wait time

Description

Halts execution of the program and continues it after a wait time. The wait time is specified in seconds.

WAIT SEC triggers an advance run stop.

Syntax

`WAIT SEC` *Wait time*

Explanation of the syntax

Element	Description
Wait time	Type: INT, REAL Number of seconds for which program execution is to be interrupted. If the value is negative, the program does not wait. With small wait times, the accuracy is determined by a multiple of 12 ms.

Examples

Interruption of program execution for 17.156 seconds:

```
WAIT SEC 17.156
```

Interruption of program execution in accordance with the variable value of V_WAIT in seconds:

```
WAIT SEC V_ZEIT
```

11.7.13 WHILE ... ENDWHILE: programming a rejecting loop

Description

Rejecting loop. Loop that is repeated as long as a certain condition is fulfilled.

If the condition is not met, program execution is resumed at the first statement after the ENDWHILE line. The condition is checked before each loop execution. If the condition is not already fulfilled beforehand, the statement block is not executed.

Loops can be nested. In the case of nested loops, the outer loop is executed completely first. The inner loop is then executed completely.

Syntax

`WHILE` *Repetition condition*

Statement block

`ENDWHILE`

Explanation of the syntax

Element	Description
Repetition condition	Type: BOOL Possible: • Variable of type BOOL • Function of type BOOL • Logic operation, e.g. a comparison, with a result of type BOOL

Example 1

The loop is executed 99 times. After execution of the loop, `W` has the value 100.

```
W=1
WHILE W<100
    W=W+1
```

```
ENDWHILE
```

Example 2

The loop is executed as long as `$IN[1]` is true.

```
WHILE $IN[1]==TRUE
    W=W+1
ENDWHILE
```

11.8 Inputs/outputs

11.8.1 ANIN: cyclically reading an analog input

Description

Cyclical reading (every 12 ms) of an analog input. ANIN triggers an advance run stop.

The robot controller has 32 analog inputs ($ANIN[1] … $ANIN[32]).

- A maximum of three ANIN ON statements can be used at the same time.
- A maximum of two ANIN ON statements can use the same variable *Value* or access the same analog input.
- All of the variables used in an ANIN statement must be declared in data lists (locally or in $CONFIG.DAT).

$ANIN[…] indicates the input voltage, adapted to the range between -1.0 and +1.0. The actual voltage depends on the settings of the analog module.

Syntax

Starting cyclical reading:

`ANIN ON` *Value* `=` *Factor* `*` *Signal name* `*` <±*Offset*>

Ending cyclical reading:

`ANIN OFF` *Signal name*

Explanation of the syntax

Element	Description
Value	Type: REAL The result of the cyclical reading is stored in *Value*. *Value* can be a variable or a signal name for an output.
Factor	Type: REAL Any factor. It can be a constant, variable or signal name.
Signal name	Type: REAL Specifies the analog input. *Signal name* must first have been declared with `SIGNAL` . It is not possible to specify the analog input $ANIN[x] directly instead of the signal name.
Offset	Type: REAL It can be a constant, variable or signal name.

Example

In this example, the REAL variable MY_VAR is written to via the analog input $ANIN[1].

$ANIN[1] must first be linked to a freely selected signal name, in this case SIGNAL_1, in the declaration section.

```
SIGNAL SIGNAL_1 $ANIN[1]
REAL MY_VAR
...
ANIN ON MY_VAR = 1.0 * SIGNAL_1
```

The cyclical scanning of `SIGNAL_1` is ended using the `ANIN OFF` statement.

```
ANIN OFF SIGNAL_1
```

11.8.2 ANOUT: writing cyclically to an analog output

Description

Cyclical writing (every 12 ms) to an analog output. ANOUT triggers an advance run stop.

The robot controller has 32 analog outputs ($ANOUT[1] … $ANOUT[32]).

- A maximum of four ANOUT ON statements can be used at the same time.
- All of the variables used in an ANOUT statement must be declared in data lists (locally or in $CONFIG.DAT).

$ANOUT[…] can have values from -1.0 to +1.0 written to it. The voltage actually generated depends on the settings of the analog module. If an attempt is made to set voltages outside the range of values, the robot controller displays the following message: *Limit {Signal name}*

Syntax

Starting cyclical writing:

`ANOUT ON` *Signal name* `=` *Factor* `*` *Control element* `<`*±Offset*`>` `<DELAY =` *±Time*`>` `<MINIMUM =` *Minimum value*`>` `<MAXIMUM =` *Maximum value*`>`

Ending cyclical writing:

`ANOUT OFF` *Signal name*

Explanation of the syntax

Element	Description
Signal name	Type: REAL Specifies the analog output. *Signal name* must first have been declared with SIGNAL . It is not possible to specify the analog output $ANOUT[x] directly instead of the signal name.
Factor	Type: REAL Any factor. It can be a constant, variable or signal name.
Control element	Type: REAL It can be a constant, variable or signal name.

Element	Description
Offset	Type: REAL It can be a constant, variable or signal name.
Time	Type: REAL Unit: seconds. By using the keyword DELAY and entering a positive or negative amount of time, the output signal can be delayed (+) or set early (-).
Minimum value, Maximum value	Type: REAL Minimum and/or maximum value to be present at the output. The actual value does not fall below/exceed these values, even if the calculated values fall outside this range. Permissible values: -1.0 to +1.0 It can be a constant, variable, structure component or array element. The minimum value must always be less than the maximum value. The sequence of the keywords MINIMUM and MAXIMUM must be observed.

Example

In this example, the output $ANOUT[5] controls the adhesive output.

A freely selected name, in this case GLUE, is assigned to the analog output in the declaration section. The amount of adhesive is to be dependent on the current path velocity (= system variable $VEL_ACT). Furthermore, the output signal is to be generated 0.5 seconds early. The minimum voltage is to be 3 V. (The voltage of the module used ranges from +10 V to -10 V.)

```
SIGNAL GLUE $ANOUT[5]
...
ANOUT ON GLUE = 0.5 * $VEL_ACT DELAY=-0.5 MINIMUM=0.30
```

The cyclical analog output is ended by using ANOUT OFF:

```
ANOUT OFF GLUE
```

11.8.3 PULSE: setting a pulse output

Description

Sets a pulse. The output is set to a defined level for a specified duration. The output is then reset automatically by the system. The output is set and reset irrespective of the previous level of the output.

At any one time, pulses may be set at a maximum of 16 outputs.

If PULSE is programmed before the first motion block, the pulse duration also elapses if the Start key is released again and the robot has not yet reached the BCO position.

The PULSE statement triggers an advance run stop. It is only executed concurrently with robot motion if it is used in a TRIGGER statement.

The pulse is not terminated in the event of an EMERGENCY STOP, an operator stop or an error stop!

Syntax

`PULSE` (*Signal, Level, Pulse duration*)

Explanation of the syntax

Element	Description
Signal	Type: BOOL Output to which the pulse is to be fed. The following are permitted: • OUT[No] • Signal variable
Level	Type: BOOL Logic expression: • TRUE represents a positive pulse (high). • FALSE represents a negative pulse (low).
Pulse duration	Type: REAL Range of values: 0.1 to 3.0 seconds. Pulse durations outside this range trigger a program stop. Pulse interval: 0.1 seconds, i.e. the pulse duration is rounded up or down. The PULSE statement is executed in the controller at the low-priority clock rate. This results in a tolerance in the order of the pulse interval (0.1 seconds). The time deviation is about 1% - 2% on average. The deviation is about 13% for very short pulses.

$OUT+PULSE

If an output is already set before the pulse, it will be reset by the falling edge of the pulse.

```
$OUT[50] = TRUE
PULSE($OUT[50],TRUE,0.5)
```

Actual pulse characteristic at output 50:

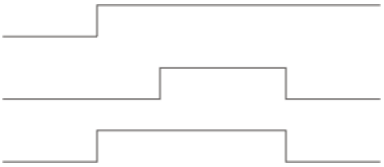


Fig. 11-1: $OUT+PULSE, example 1

If a negative pulse is applied to an output that is set to Low, the output remains Low until the end of the pulse and is then set to High:

```
$OUT[50] = FALSE
PULSE($OUT[50],FALSE,0.5)
```

Actual pulse characteristic at output 50:

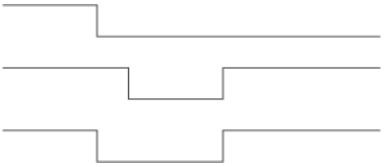


Fig. 11-2: $OUT+PULSE, example 2

PULSE+$OUT

If the same output is set during the pulse duration, it will be reset by the falling edge of the pulse.

```
PULSE($OUT[50],TRUE,0.5)
$OUT[50] = TRUE
```

Actual pulse characteristic at output 50:

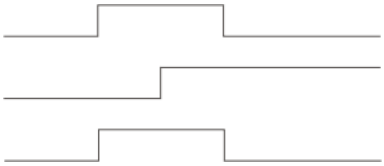


Fig. 11-3: PULSE+$OUT, example 1

If the output is reset during the pulse duration, the pulse duration is reduced accordingly:

```
PULSE($OUT[50],TRUE,0.5)
$OUT[50] = FALSE
```

Actual pulse characteristic at output 50:

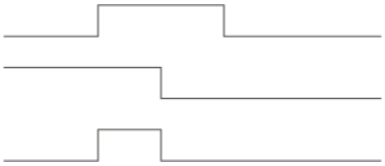


Fig. 11-4: PULSE+$OUT, example 2

If an output is set to FALSE during a pulse and then back to TRUE, the pulse is interrupted and then resumed when the output is set to TRUE. The overall duration from the first rising edge to the last falling edge (i.e. including the duration of the interruption) corresponds to the duration specified in the PULSE statement.

```
PULSE($OUT[50],TRUE,0.8)
$OUT[50]=FALSE
$OUT[50]=TRUE
```

Actual pulse characteristic at output 50:

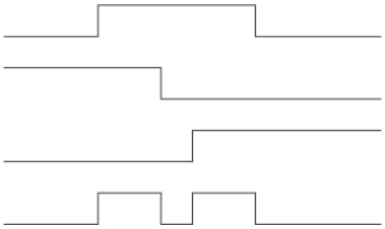


Fig. 11-5: PULSE+$OUT, example 3

The actual pulse characteristic is only specified as above if $OUT[x]=TRUE is set during the pulse. If $OUT[x]=TRUE is not set until after the pulse (see line 3), then the actual pulse characteristic is as follows (line 4):

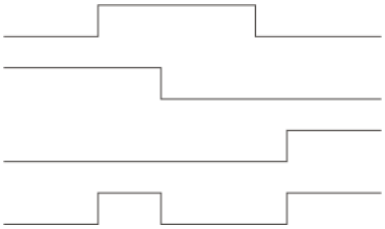


Fig. 11-6: PULSE+$OUT, example 4

PULSE+PULSE

If several PULSE statements overlap, it is always the last PULSE statement that determines the end of the overall pulse duration.

If a pulse is activated again before the falling edge, the duration of the second pulse starts at this moment. The overall pulse duration is thus shorter than the sum of the values of the first and second pulses:

```
PULSE($OUT[50],TRUE,0.5)
PULSE($OUT[50],TRUE,0.5)
```

Actual pulse characteristic at output 50:

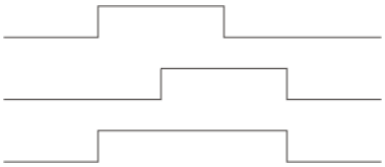


Fig. 11-7: PULSE+PULSE, example 1

If, during the pulse duration of a positive pulse, a negative pulse is sent to the same output, only the second pulse is taken into consideration from this moment onwards:

```
PULSE($OUT[50],TRUE,0.5)
PULSE($OUT[50],FALSE,0.5)
```

Actual pulse characteristic at output 50:

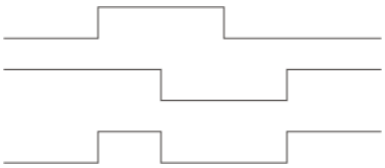


Fig. 11-8: PULSE+PULSE, example 2

```
PULSE($OUT[50],TRUE,3.0)
PULSE($OUT[50],FALSE,1.0)
```

Actual pulse characteristic at output 50:

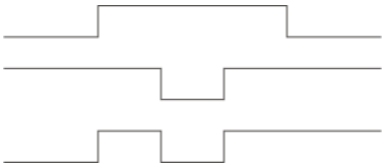


Fig. 11-9: PULSE+PULSE, example 3

PULSE+END

If a pulse is programmed before the END statement, the duration of program execution is increased accordingly.

```
PULSE($OUT[50],TRUE,0.8)
END
```

Program active

Actual pulse characteristic at output 50:

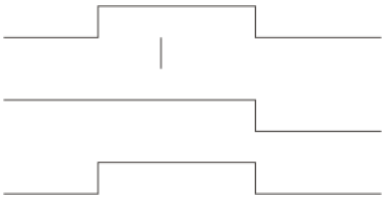


Fig. 11-10: PULSE+END, example

PULSE+RESET/CANCEL

If program execution is reset (RESET) or aborted (CANCEL) while a pulse is active, the pulse is immediately reset:

`PULSE($OUT[50],TRUE,0.8)`

`RESET` or `CANCEL`

Actual pulse characteristic at output 50:

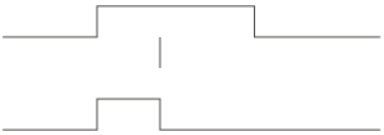


Fig. 11-11: PULSE+RESET, example

11.8.4 SIGNAL: signal declaration for inputs/outputs

Description

SIGNAL links predefined signal variables for inputs or outputs with a name.

Such a link, i.e. a SIGNAL declaration, is required in order to be able to address an analog input or output. An input or output may appear in several SIGNAL declarations.

The user can declare signals in the following files:

- In DAT files, in the section EXTERNAL DECLARATIONS
- In SRC files, in the declaration section
- In $CONFIG.DAT, in the section USER GLOBALS

There are also SIGNAL declarations that are predefined in the system. They can be found in the file $machine.DAT in the directory KRC:\STEU \MADA. These declarations can be deactivated in $machine.DAT using the keyword FALSE.

Syntax

Declaration of signal names for inputs and outputs:

`<GLOBAL> SIGNAL` *Signal name Signal variable* `<TO` *Signal variable*`>`

Deactivation of a predefined SIGNAL declaration:

`SIGNAL` *System signal name* `FALSE`

Explanation of the syntax

Element	Description
`GLOBAL`	Only possible for signals defined in a DAT file. (>>> *11.3.4 "Areas of validity" Page 453*)
Signal name	Any name

Element	Description
Signal variable	Predefined signal variable. The following types are available: • $IN[x] • $OUT[x] • $ANIN[x] • $ANOUT[x]
`TO`	Groups together several consecutive binary inputs or outputs (max. 32) to form a digital input or output. The combined signals can be addressed with a decimal name, a hexadecimal name (prefix H) or with a bit pattern name (prefix B). They can also be processed with Boolean operators.
System signal name	Signal name predefined in the system, e.g. `$T1`.
`FALSE`	Deactivates a SIGNAL declaration predefined in the system. The inputs or outputs to which the SIGNAL declaration refers are thus available again for other purposes. FALSE is not a Boolean value here, but a keyword. The option TRUE is not available. If the SIGNAL declaration that has been deactivated by means of FALSE is to be reactivated, the program line containing the entry FALSE must be deleted.

Example 1

The output $OUT[7] is assigned the name `START_PROCESS`. The output $OUT[7] is set.

```
SIGNAL START_PROCESS $OUT[7]
START_PROCESS = TRUE
```

Example 2

The outputs $OUT[1] to $OUT[8] are combined to form one digital output under the name `OUTWORT`. The outputs $OUT[3], $OUT[4], $OUT[5] and $OUT[7] are set.

```
SIGNAL OUTWORT $OUT[1] TO $OUT[8]
OUTWORT = 'B01011100'
```

11.9 Subprograms and functions

11.9.1 Calling a subprogram

Description

Subprograms are programs which are accessed by means of branches from the main program. Once the subprogram has been executed, the main program is resumed from the line directly after the subprogram call.

- **Local subprograms** are contained in the same SRC file as the main program. They can be made to be recognized globally using the keyword GLOBAL.
- **Global subprograms** are programs with a separate SRC file of their own, which is accessed from another program by means of a branch.

A subprogram is called in the main program by specifying the name of the subprogram followed by round brackets.

Example

In the following example, the subprogram `my_subprogram` is called:

```
my_subprogram()
```

11.9.2 Calling a function

Description

A function is a subprogram that returns a certain value to the main program. Functions have a data type.

A function is called in a similar way to a subprogram: specify the name of the function in the main program followed by round brackets. A function call can never stand alone, however; instead, the value must constantly be assigned to a variable of the same data type.

Example

Examples of calls from the main program:

```
REALVAR = REALFUNCTION()
```

```
INTVAR = 5 * INTFUNCTION() + 1
```

11.9.3 DEFFCT ... ENDFCT: defining a function

Syntax

`DEFFCT` *Data type Name* `(`<*Variable*`:IN |OUT>)`

<*Statements*>

`RETURN` *Function value*

`ENDFCT`

Explanation of the syntax

Element	Description
Data type	Data type of the function
Name	Name of the function
Variable	If a value is transferred in the function: name of the variable to which the value is transferred (>>> *11.9.5 "Transferring parameters to a subprogram or function" Page 516*)
`IN` | `OUT`	If a value is transferred in the function: type of transfer
Function value	(>>> *11.9.4 "RETURN: jump back to the calling program" Page 515*)

Example

(>>> *11.9.4 "RETURN: jump back to the calling program" Page 515*)

11.9.4 RETURN: jump back to the calling program

Description

Jump from a subprogram or function back to the program from which the subprogram or function was called.

Subprograms

RETURN can be used to return to the main program if a certain condition is met in the subprogram. No values from the subprogram can be transferred to the main program.

Functions

Functions must be ended by a RETURN statement containing the value that has been determined. The determined value is hereby transferred to the program from which the function was called.

Syntax

For subprograms:

`RETURN`

For functions:

`RETURN` *Function value*

Explanation of the syntax

Element	Description
Function value	Type: The data type of *Function_Value* must match the data type of the function. *Function_Value* is the value determined by the function. The value can be specified as a constant, a variable or an expression.

Example 1

Return from a subprogram to the program from which it was called, dependent on a condition.

```
DEF PROG_2()
    ...
    IF $IN[5]==TRUE THEN
    RETURN
    ...
END
```

Example 2

Return from a function to the program from which it was called. The value `X` is transferred.

```
DEFFCT INT CALCULATE(X:IN)
    INT X
    X=X*X
    RETURN X
ENDFCT
```

11.9.5 Transferring parameters to a subprogram or function

Description

Parameters can be transferred from a main program to local and global subprograms and functions.

There are 2 ways of transferring parameters:

- As IN parameters

  The value of the variable remains unchanged in the main program.

  This transfer type is also called “Call by Value”.
- As OUT parameters

  The subprogram reads the value, modifies it and writes the new value back to the main program.

  This transfer type is also called “Call by Reference”.

Recommendation: Always transfer a parameter to a variable of the same data type.
It is possible to transfer parameters to a different data type, but with certain restrictions.
(>>> *11.9.6 "Transferring a parameter to a different data type" Page 520*)

Example 1

Transferring parameters to a local subprogram:

```
1  DEF MY_PROG( )
2  DECL REAL r,s
3  ...
4  CALC_1(r)
5  ...
6  CALC_2(s)
7  ...
8  END
_____________________________________
9  DEF CALC_1(num1:IN)
10 DECL REAL num1
11 ...
12 END
_____________________________________
13 DEF CALC_2(num2:OUT)
14 DECL REAL num2
15 ...
16 END
```

Line	Description
4	The subprogram CALC_1 is called and the parameter “r” is transferred.
6	The subprogram CALC_2 is called and the parameter “s” is transferred.
9	`num1`: The name of the variable to which the value of “r” is transferred. `IN` means: “r” is only transferred for reading.
10, 14	The variables to which values are transferred must be declared.

Line	Description
13	`num2`: The name of the variable to which the value of “s” is transferred. `OUT` means: “s” is transferred, modified and written back to the main program.

Example 2

Transferring parameters to a global function:

Main program MY_PROG( ):

```
1 DEF MY_PROG( )
2 DECL REAL result, value
3 value = 2.0
4 result = CALC(value)
5 ...
...
END
```

Line	Description
3	“value” is assigned the value “2.0”.
4	The function CALC is called and the value of “value” is transferred. The return value of the function is assigned to the variable “result”.

What happens if the value is transferred as an IN parameter?

CALC() function with IN:

```
1 DEFFCT REAL CALC(num:IN)
2 DECL REAL return_value, num
3 num = num + 8.0
4 return_value = num * 100.0
5 RETURN(return_value)
6 ENDFCT
```

Line	Description
1	The value of “value” is transferred to “num” as an IN parameter. The value is still 2.0.
3	The value of “num” is modified. The value is now 10.0.
4, 5	The value of “return_value” is calculated and returned to the variable “result” in the main program. The value is 1 000.0.
6	The function is terminated and execution of the main program is resumed from line 5. **Note:** The value of “value” in the main program is unchanged: 2.0.

What happens if the value is transferred as an OUT parameter?

CALC() function with OUT:

```
1 DEFFCT REAL CALC(num:OUT)
2 DECL REAL return_value, num
3 num = num + 8.0
4 return_value = num * 100.0
```

```
5 RETURN(return_value)
6 ENDFCT
```

Line	Description
1	The value of “value” is transferred to “num” as an OUT parameter. The value is still 2.0.
3	The value of “num” is modified. The value is now 10.0.
4, 5	The value of “return_value” is calculated and returned to the variable “result” in the main program. The value is 1 000.0.
6	The function is terminated and execution of the main program is resumed from line 5. **Note:** The value of “value” in the main program is now 10.0.

Transferring multiple parameters

Transferring multiple parameters:

The sequence automatically determines which parameter is transferred to which parameter: The first parameter is transferred to the first parameter in the subprogram, the second to the second parameter in the subprogram, etc.

```
1 DEF MY_PROG( )
2 DECL REAL w
3 DECL INT a, b
4 ...
5 CALC(w, b, a)
6 ...
7 CALC(w, 30, a)
8 ...
9 END
_____________________________________
10 DEF CALC(ww:OUT, bb:IN, oo:OUT)
11 DECL REAL ww
12 DECL INT oo, bb
13 ...
14 END
```

Line	Description
5	“w” is transferred to “ww” as an OUT parameter. “b” is transferred to “bb” as an IN parameter. “a” is transferred to “oo” as an OUT parameter.
7	“w” is transferred to “ww” as an OUT parameter. “30” is transferred to “bb” as an IN parameter. “a” is transferred to “oo” as an OUT parameter.

It is also possible not to transfer a value to a receiving variable in the subprogram, provided that this value is not required in the subprogram. This makes it easier to adapt the program to changing sequences. Example: CALC (w, ,a)

Transferring arrays

Transferring arrays:

- Arrays may only be transferred as OUT parameters.
  Exception: CHAR arrays can also be transferred as IN parameters.
- Only complete arrays can be transferred to another array.
- Always declare the array in the subprogram without an array size. The array size adapts itself to the original array.

```
1  DEF MY_PROG( )
2  DECL CHAR name[10]
3  ...
4  name="OKAY"
5  CALC(name[])
6  ...
7  END
_____________________________________
8  DEF CALC(my_name[]:OUT)
9  DECL CHAR my_name[]
10 ...
11 END
```

Line	Description
5, 8	Only complete arrays can be transferred to another array.
8	Arrays may only be transferred as OUT parameters.
9	Always declare the array in the subprogram without an array size. The array size adapts itself to the original array.

In the case of transferring multidimensional arrays, array sizes are again not specified. However, the dimension of the array must be specified by means of commas.

Examples:

`ARRAY_1D[]` (1-dimensional)

`ARRAY_2D[,]` (2-dimensional)

`ARRAY_3D[,,]` (3-dimensional)

Transferring individual array elements:

An individual array element may only be transferred to a variable, not to an array.

```
1  DEF MY_PROG( )
2  DECL CHAR name[10]
3  ...
4  name="OKAY"
5  CALC(name[1])
6  ...
7  END
_____________________________________
8  DEF CALC(symbol:IN)
9  DECL CHAR symbol
10 ...
11 END
```

Line	Description
2	A CHAR array with 10 elements is declared.
4	Values are assigned to the first 4 elements of the array. This corresponds to: name[1] = "O" name[2] = "K" name[3] = "A" name[4] = "Y" (A CHAR variable can only ever contain 1 ASCII character.)
5	The subprogram CALC is called and the value of the first element is transferred, i.e. the value “O”.
8	Individual array elements can also be transferred as IN parameters.
9	The variable to which the value of the array element is transferred must be declared (a variable, not an array).

11.9.6 Transferring a parameter to a different data type

It is always possible to transfer a value to the same data type. The following applies for transfer to a different data type:

Type in the main program	Type in the subprogram	Effect
BOOL	INT, REAL, CHAR	Transfer not possible; error message
INT, REAL, CHAR	BOOL	
INT	REAL	INT value is used as REAL value.
INT	CHAR	Character from the ASCII table is used.
CHAR	INT	INT value from the ASCII table is used.
CHAR	REAL	REAL value from the ASCII table is used.
REAL	INT	REAL values are rounded.
REAL	CHAR	REAL values are rounded; character from the ASCII table is used.

11.10 Interrupt programming

11.10.1 INTERRUPT … DECL … WHEN … DO … : declaring an interrupt

Description

In the case of a defined event, e.g. an input, the controller interrupts the current program and executes a defined subprogram. The event and the subprogram are defined by INTERRUPT … DECL … WHEN … DO ….

Once the subprogram has been executed, the interrupted program is resumed at the point at which it was interrupted. Exception: RESUME.

A subprogram called by an interrupt is called an interrupt program.

A maximum of 64 interrupts may be declared simultaneously. An interrupt declaration may be overwritten by another at any time.

An interrupt can optionally be declared with BRAKE. The BRAKE statement is then executed without a delay as soon as the declared interrupt is

detected. This means that the braking process has already been initiated when the interrupt program is started.

The interrupt declaration is an instruction. It must be situated in the instruction section of the program and not in the declaration section.

Following the declaration, an interrupt is initially inactive. The interrupt must be activated before it can respond to the defined event.
(>>> *11.10.2 "INTERRUPT ON/OFF: activating or deactivating an interrupt" Page 523*)

Interrupt programs must not contain any spline motions.

Syntax

`<GLOBAL> INTERRUPT <WITH BRAKE <F|FF>> DECL` *Prio* `WHEN` *Event* `DO` *Subprogram*

Explanation of the syntax

Element	Description
`GLOBAL`	An interrupt is only recognized at, or below, the level in which it is declared. In other words, an interrupt declared in a subprogram is not recognized in the main program (and cannot be activated there). If an interrupt is also to be recognized at higher levels, the declaration must be preceded by the keyword GLOBAL.
`WITH BRAKE <F|FF>`	When the interrupt is detected, a BRAKE statement is executed. (>>> *11.10.4 "BRAKE: stopping the robot from an interrupt program" Page 526*)
Prio	Type: INT If several interrupts occur at the same time, the interrupt with the highest priority is processed first, then those of lower priority. 1 = highest priority. Priorities 1, 2, 4 to 39 and 81 to 128 are available. **Note:** Priorities 3 and 40 to 80 are reserved for use by the system. They must not be used by the user because this would cause system-internal interrupts to be overwritten and result in errors.
Event	Type: BOOL Event that is to trigger the interrupt. Structure components are impermissible. The following are permitted: • Global Boolean variables • Signal names • Comparisons • Simple logic operations: NOT, OR, AND or EXOR
Subprogram	The name of the interrupt program to be executed. Runtime variables must not be transferred to the interrupt program as parameters.

Example 1

Declaration of an interrupt with priority 23 that calls the subprogram UP1 if $IN[12] is true. The parameters 20 and VALUE are transferred to the subprogram.

```
INTERRUPT DECL 23 WHEN $IN[12]==TRUE DO UP1(20,VALUE)
```

Example 2

Two objects, the positions of which are detected by two sensors connected to inputs 6 and 7, are located on a programmed path. The robot is to be moved subsequently to these two positions.

For this purpose, the two detected positions are saved as points P_1 and P_2. These points are then addressed in the second section of the main program.

If the robot controller detects an event defined by means of INTERRUPT … DECL … WHEN … DO …, it always saves the current robot position in the system variables $AXIS_INT (axis-specific) and $POS_INT (Cartesian).

Main program:

```
DEF PROG()
...
INTERRUPT DECL 10 WHEN $IN[6]==TRUE DO UP1()
INTERRUPT DECL 20 WHEN $IN[7]==TRUE DO UP2()
...
INTERRUPT ON
LIN START
LIN END
INTERRUPT OFF
LIN P_1
LIN P_2
...
END
```

Local interrupt program 1:

```
DEF UP1()
P_1=$POS_INT
END
```

Local interrupt program 2:

```
DEF UP2()
P_2=$POS_INT
END
```

Example 3

If a collision is detected, the robot must be stopped quickly and reliably. For this purpose, an interrupt is declared with BRAKE F.

If the positive torque in axis A1 exceeds 1500 Nm, the interrupt immediately initiates a BRAKE F and calls the subprogram STOP_FAST(). When the robot is stationary, a return motion is carried out to the Cartesian position $POS_INT, offset by -10 mm in tool direction X. $POS_INT is the position at which the interrupt was triggered.

```
DEF PROG()
...
```

```
INTERRUPT WITH BRAKE F DECL 25 WHEN $TORQUE_AXIS_ACT[1]>1500
DO STOP_FAST()
INTERRUPT ON 25
...
END
...
DEF STOP_FAST()
BRAKE F
PTP $POS_INT:{x -10}
...
END
```

11.10.2 INTERRUPT ON/OFF: activating or deactivating an interrupt

Description

This statement activates or deactivates an interrupt.

Following declaration, an interrupt is initially inactive. The interrupt must be activated before it can respond to the defined event.

Syntax

`INTERRUPT` *Action* <*Number*>

Explanation of the syntax

Element	Description
Action	• `ON`: Activates an interrupt. • `OFF`: Deactivates an interrupt.
Number	Type: INT Number (= priority) of the interrupt to which the *Action* is to refer. *Number* can be omitted. In this case, `ON` or `OFF` refers to all declared interrupts.

Up to 32 interrupts may be active at any one time. In this regard, particular attention must be paid to the following:

- If, in the case of `INTERRUPT ON`, the *Number* is omitted, all declared interrupts are activated. The maximum permissible total of 32 may not be exceeded, however.
- If a trigger calls a subprogram, it counts as an active interrupt until the subprogram has been executed.

If, in the interrupt declaration, a Boolean variable, e.g. an input, has been defined as the *Event*:

- In this case, the interrupt is triggered by a change of state, e.g. for $IN[x]==TRUE by the change from FALSE to TRUE. The state must therefore not already be present at INTERRUPT ON, as the interrupt is not then triggered.

- Furthermore, the following must also be considered in this case: the change of state must not occur until at least one interpolation cycle after INTERRUPT ON.

  (This can be achieved by programming a WAIT SEC 0.012 after INTERRUPT ON. If no advance run stop is desired, a CONTINUE command can also be programmed before the WAIT SEC.)

  The reason for this is that INTERRUPT ON requires one interpolation cycle (= 12 ms) before the interrupt is actually activated. If the state changes before this, the interrupt cannot detect the change.

Example 1

The interrupt with priority 2 is activated. (The interrupt must already be declared.)

```
INTERRUPT ON 2
```

Example 2

A non-path-maintaining EMERGENCY STOP is executed via the hardware during application of adhesive. The application of adhesive is stopped by the program and the adhesive gun is repositioned onto the path after enabling (by input 10).

```
DEF PROG()
...
INTERRUPT DECL 1 WHEN $STOPMESS DO STOP_PROG()
LIN P_1
INTERRUPT ON
LIN P_2
INTERRUPT OFF
...
END
```

```
DEF STOP_PROG()
BRAKE F
GLUE=FALSE
WAIT FOR $IN[10]
LIN $POS_RET
GLUE=TRUE
END
```

11.10.3 INTERRUPT DISABLE/ENABLE: disabling or enabling an interrupt

Description

This statement disables an active interrupt or re-enables a disabled interrupt.

Behavior:

A disabled interrupt does not respond when the defined event occurs, i.e. it does not call the defined subprogram. The robot controller notices, however, that the event has occurred. Once the interrupt is re-enabled, it responds retroactively to the event. (>>> *"Example 1" Page 525*)

- The time that elapses before the interrupt is re-enabled may vary. The response to an event may therefore be greatly delayed.
- If, by the time the interrupt is enabled, the event has occurred repeatedly, the interrupt nevertheless only responds once.

- If, by the time the interrupt is enabled, the event has reversed again, the interrupt responds regardless. The decisive factor is that the event has occurred (at least) once. (>>> *"Example 2" Page 526*)

If a disabled interrupt is no longer enabled, but rather is deactivated (i.e. INTERRUPT OFF follows INTERRUPT DISABLE), this interrupt deletes the flag for the event. If the interrupt is then re-enabled, it responds only when there is a new event.

Interrupts with the same event do not influence one another:

If the same event is defined for several interrupts, the interrupts do not influence each other. The robot controller remembers events separately for every interrupt and deletes the flags separately for every interrupt.

Syntax

`INTERRUPT` *Action* <*Number*>

Explanation of the syntax

Element	Description
Action	• `DISABLE`: Disables an active interrupt. • `ENABLE`: Enables a disabled interrupt.
Number	Type: INT Number (= priority) of the interrupt to which the *Action* is to refer. *Number* can be omitted. In this case, `DISABLE` or `ENABLE` refers to all active interrupts.

If `DISABLE` *Number* or `ENABLE` *Number* is applied to an inactive interrupt, this is impermissible and the robot controller generates an error message.

Example 1

The precondition for the sequence shown in the illustration is that the interrupt has been declared and is active:

```
INTERRUPT DECL 1 WHEN $IN[7]==TRUE DO UP1()
...
INTERRUPT ON 1
```

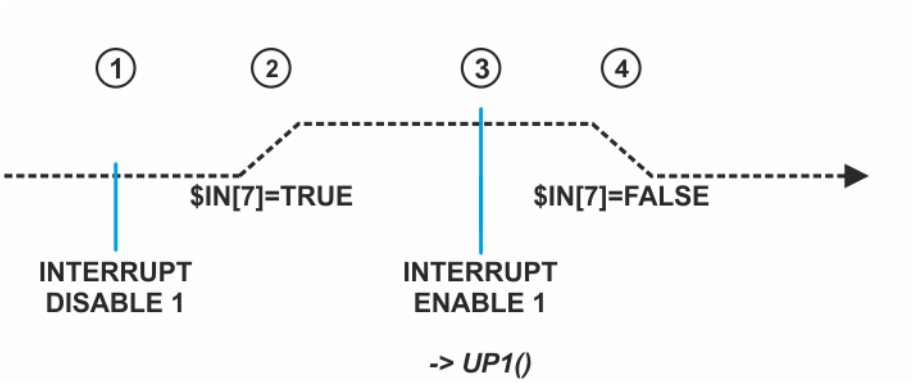


Fig. 11-12: Example 1

Item	Description
1	The interrupt is disabled.
2	The defined event occurs: input 7 is set to TRUE. The interrupt does not respond, as it is disabled.
3	The interrupt is re-enabled. It responds and calls the defined subprogram.
4	Input 7 is set back to FALSE.

Example 2

The precondition for the sequence shown in the illustration is that the interrupt has been declared and is active:

```
INTERRUPT DECL 1 WHEN $IN[7]==TRUE DO UP1()
...
INTERRUPT ON 1
```

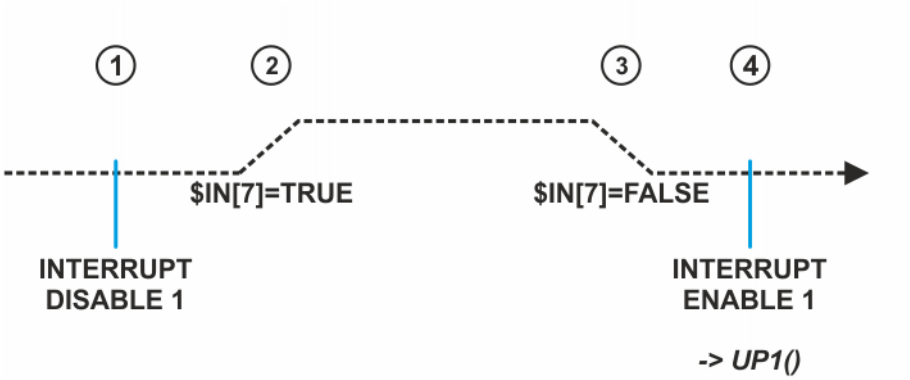


Fig. 11-13: Example 2

Item	Description
1	The interrupt is disabled.
2	The defined event occurs: input 7 is set to TRUE. The interrupt does not respond, as it is disabled.
3	Input 7 is set back to FALSE.
4	The interrupt is re-enabled. It responds and calls the defined subprogram. This is despite the fact that $IN[7] is no longer TRUE. The decisive factor is that the defined event occurred (at least) once while the interrupt was disabled.

11.10.4 BRAKE: stopping the robot from an interrupt program

Description

BRAKE stops the robot.

BRAKE may only be used in an interrupt program or in an interrupt declaration.

Using BRAKE in the interrupt declaration has the advantage that the braking reaction is initiated without a delay as soon as the interrupt is detected. The system does not wait until the interrupt program has been started and the line with the BRAKE statement has been processed before initiat-

ing the braking reaction. This means that the braking reaction is not dependent on the interpreter.

> If the BRAKE statement is present in the interrupt declaration, it can be repeated in the interrupt program. This is not necessary, but increases the legibility of the program.

The interrupt program is not continued until the robot has come to a stop. The robot motion is resumed as soon as the interrupt program has been completed.

Syntax

```
BRAKE <F|FF>
```

Explanation of the syntax

Element	Description
BRAKE	In the case of a BRAKE statement without F, the robot brakes with ramp-down (path-maintaining) braking. The synchronous robot axes are braked slowly by means of continuous override reduction and the programmed path is not left. Application: Gentle stopping from high speeds
BRAKE F	In the case of a BRAKE statement with F, the robot brakes in the same way as for a path-maintaining EMERGENCY STOP. The synchronous robot axes are braked as quickly as possible without leaving the programmed path. Application: Rapid stopping from high speeds
BRAKE FF	In the case of a BRAKE statement with FF, the robot brakes with maximum braking (not path-maintaining). All synchronous and asynchronous robot axes are stopped and the programmed path is left. In the low velocity range, a particularly short braking distance can be achieved. Application: Very rapid stopping from low speeds

Examples

BRAKE in interrupt declaration:

BRAKE in interrupt program: (>>> *11.10.2 "INTERRUPT ON/OFF: activating or deactivating an interrupt" Page 523*)

11.10.5 RESUME: cancelling interrupt programs

Description

RESUME cancels all running interrupt programs and subprograms up to the level at which the current interrupt was declared.

RESUME may only occur in interrupt programs. (Not in interrupt programs, however, that are called by an interrupt that is declared as GLOBAL.) When the RESUME statement is activated, the advance run pointer must not be at the level where the interrupt was declared, but at least one level lower.

Changing the variable $BASE in the interrupt program only has an effect there. The computer advance run, i.e. the variable $ADVANCE, must not be modified in the interrupt program.

Response following RESUME

Please note the following regarding the behavior of the robot controller after a RESUME:

- **SLIN, SPTP, LIN, PTP**:

  Following a RESUME statement, these motions are carried out as programmed. (No change in behavior.)
- **CIRC**:

  If the first motion after RESUME is a CIRC motion, this is executed as LIN.
- **SCIRC, spline block**:

  If the first motion after RESUME is an SCIRC or a spline block, the program then stops with the following message: *SPLINE or SCIRC not allowed after Resume*.

  The program cannot be continued. The motion type after RESUME must be modified. After a block selection or a program reset, motion can be resumed.

The reason for the behavior in the case of CIRC, SCIRC and spline block: Following a RESUME statement, the robot is not situated at the original start point of the motion. This motion would thus differ from how it was originally planned; this can potentially be very dangerous in the case of CIRC, SCIRC and spline block.

After a RESUME, do not program an SCIRC or spline block.
After a RESUME, do not program a CIRC if possible.

WARNING

If a CIRC motion is programmed as the first motion after RESUME, the robot must be able to reach the end point safely, by means of a LIN motion, from any position in which it could be located when the RESUME statement is executed. Failure to take this precaution into consideration may result in death, injuries or damage to property.

Syntax

```
RESUME
```

Example

The robot is to search for a part on a path. The part is detected by means of a sensor at input 15. Once the part has been found, the robot is not to continue to the end point of the path, but to return to the interrupt position and pick up the part. The main program is then to be resumed.

Main program PROG():

```
DEF PROG()
INI
...
INTERRUPT DECL 21 WHEN $IN[15] DO FOUND()
PTP HOME
...
SEARCH()
...
END
```

Motions that are to be canceled by means of BRAKE and RESUME must be located in a subprogram. For this reason, the search path is not directly programmed in the main program, but rather in the subprogram SEARCH().

Subprogram SEARCH() with search path:

```
DEF SEARCH()
INTERRUPT ON 21
SPLINE
  SPL START_SEARCH
  SPL IN_BETWEEN
  SPL END_SEARCH
ENDSPLINE
WAIT FOR TRUE
...
END
```

When the RESUME statement is activated, the advance run pointer must not be at the level where the current interrupt was declared. To prevent this, an advance run stop is triggered here via WAIT FOR TRUE.

Interrupt program FOUND():

```
DEF FOUND()
INTERRUPT OFF 21
BRAKE
LIN $POS_INT
... ;The robot grips the found part.
RESUME
END
```

The braking process causes the robot to move slightly away from the position at which the interrupt was triggered. LIN $POS_INT causes the robot to return to the position at which the interrupt was triggered.

The motion type LIN was used here because interrupt programs must not contain any spline motions.

After LIN $POS_INT, the robot grips the part. (Not programmed in this example.)

RESUME causes the main program to be resumed after the part has been gripped. Without the RESUME statement, the subprogram SEARCH() would be resumed after END.

11.11 Path-related switching actions (=Trigger)

11.11.1 TRIGGER WHEN DISTANCE

Description

The Trigger triggers a user-defined statement. The robot controller executes the statement parallel to the robot motion.

The trigger can optionally refer to the start or end point of the motion. The statement can either be triggered directly at the reference point, or it can be shifted in time.

Syntax

`TRIGGER WHEN DISTANCE=`*Position* `DELAY=`*Time* `DO` *Statement* `<PRIO=`*Priority*`>`

Explanation of the syntax

Element	Description
Position	Type: INT; variable or constant Reference point of the trigger • **0:** Start point • **1:** End point (>>> *"Reference point for approximate positioning" Page 530*)
Time	Type: REAL; variable, constant or function; unit: ms Shift in time relative to *Position*. If no offset is desired, set *Time* = 0. • Negative value: Offset towards the start of the motion • Positive value: Offset towards the end of the motion (>>> *"Max. offset" Page 531*) If *Time* calls a function, the function is subject to constraints. (>>> *11.11.3 "Constraints for functions in the trigger" Page 540*)
Statement	Possible: • Assignment of a value to a variable **Note:** There must be no runtime variable on the left-hand side of the assignment. • OUT statement; PULSE statement; CYCFLAG statement • Subprogram call. In this case, *Priority* must be specified.
Priority	Type: INT; variable or constant Priority of the trigger. Only relevant if *Statement* calls a subprogram, and then obligatory. Priorities 1, 2, 4 to 39 and 81 to 128 are available. Priorities 40 to 80 are reserved for cases in which the priority is automatically assigned by the system. If the priority is to be assigned automatically by the system, the following is programmed: `PRIO = -1`. If several triggers call subprograms at the same time, the trigger with the highest priority is processed first, then the triggers of lower priority. 1 = highest priority.

If a trigger calls a subprogram, it counts as an active interrupt until the subprogram has been executed. Up to 32 interrupts may be active at any one time.

Reference point for approximate positioning

Where is the reference point if the start or end point is approximated?

- DISTANCE = 0:
  If the start point is approximated, the reference point lies at the end of the approximate positioning arc.
- DISTANCE = 1:
  If the end point is approximated, the reference point lies in the middle of the approximate positioning arc.

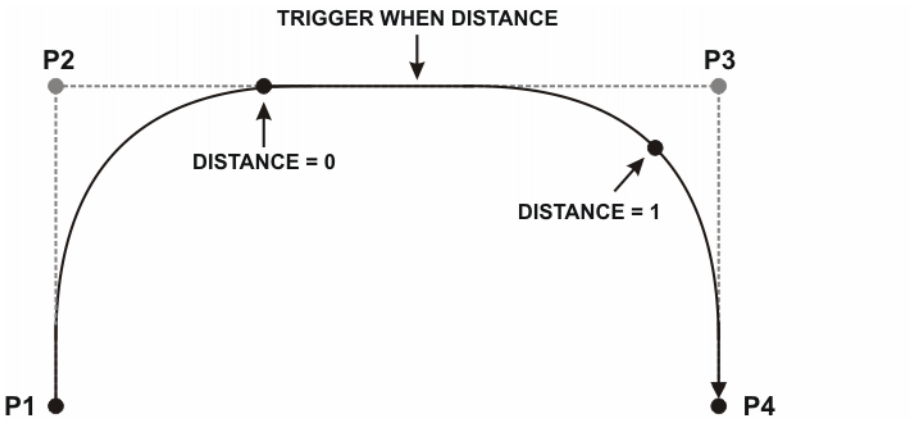


Fig. 11-14: TRIGGER WHEN DISTANCE reference point for approximate positioning

Max. offset

There are limits to the distance the reference point can be offset. The following table specifies the maximum possible offsets. If larger, and thus invalid, offsets are programmed, the robot controller switches the trigger at the permissible limit at the latest. In T1/T2, it generates a corresponding message.

DISTANCE = ...	Maximum negative offset ...	Maximum positive offset ...
DISTANCE = 0	- - - (No negative offset possible.)	• Up to end point • If the end point is approximated: up to the start of the approximate positioning arc
DISTANCE = 1 and End point = exact positioning point	• Up to start point • If the start point is approximated: up to the end of the approximate positioning arc	- - - (No positive offset possible.)
DISTANCE = 1 and End point = approximated	Up to the start of the approximate positioning arc of the end point	Up to the end of the approximate positioning arc of the end point

Example 1

130 milliseconds after P_2, $OUT[8] is set to TRUE.

```
LIN P_2
TRIGGER WHEN DISTANCE=0 DELAY=130 DO $OUT[8]=TRUE
LIN P_3
```

Example 2

In the middle of the approximate positioning arc of P_5, the subprogram MY_SUBPROG with priority 5 is called.

```
PTP P_4
TRIGGER WHEN DISTANCE=1 DELAY=0 DO MY_SUBPROG() PRIO=5
PTP P_5 C_DIS
```

```
PTP P_6
```

Example 3

Explanation of the diagram (>>> *Fig. 11-15*):

In the diagram, the approximate positions in which the Triggers would be triggered are indicated by arrows. Start, middle and end of each approximate positioning arc are marked (with **start*, **middle* and **end*).

```
1 DEF PROG()
2 ...
3 PTP P_0
4 TRIGGER WHEN DISTANCE=0 DELAY=40 DO A=12
5 TRIGGER WHEN DISTANCE=1 DELAY=-20 DO UP1() PRIO=10
6 LIN P_1
7 ...
8 TRIGGER WHEN DISTANCE=0 DELAY=10 DO UP2(A) PRIO=5
9 TRIGGER WHEN DISTANCE=1 DELAY=15 DO B=1
10 LIN P_2 C_DIS
11 ...
12 TRIGGER WHEN DISTANCE=0 DELAY=10 DO UP2(B) PRIO=12
13 TRIGGER WHEN DISTANCE=1 DELAY=0 DO UP(A,B,C) PRIO=6
14 LIN P_3 C_DIS
15 ...
16 TRIGGER WHEN DISTANCE=0 DELAY=50 DO UP2(A) PRIO=4
17 TRIGGER WHEN DISTANCE=1 DELAY=-80 DO A=0
18 LIN P_4
19 ...
20 END
```

Line	Switching ranges of the triggers
4	Switching range: P_0 to P_1
5	Switching range: P_0 to P_1
8	Switching range: P_1 to P_2*start
9	Switching range: P_2*start to P_2*end
12	Switching range: P_2*end to P_3*start
13	Switching range: P_3*start to P_3*end
16	Switching range: P_3*end to P_4
17	Switching range: P_3*end to P_4

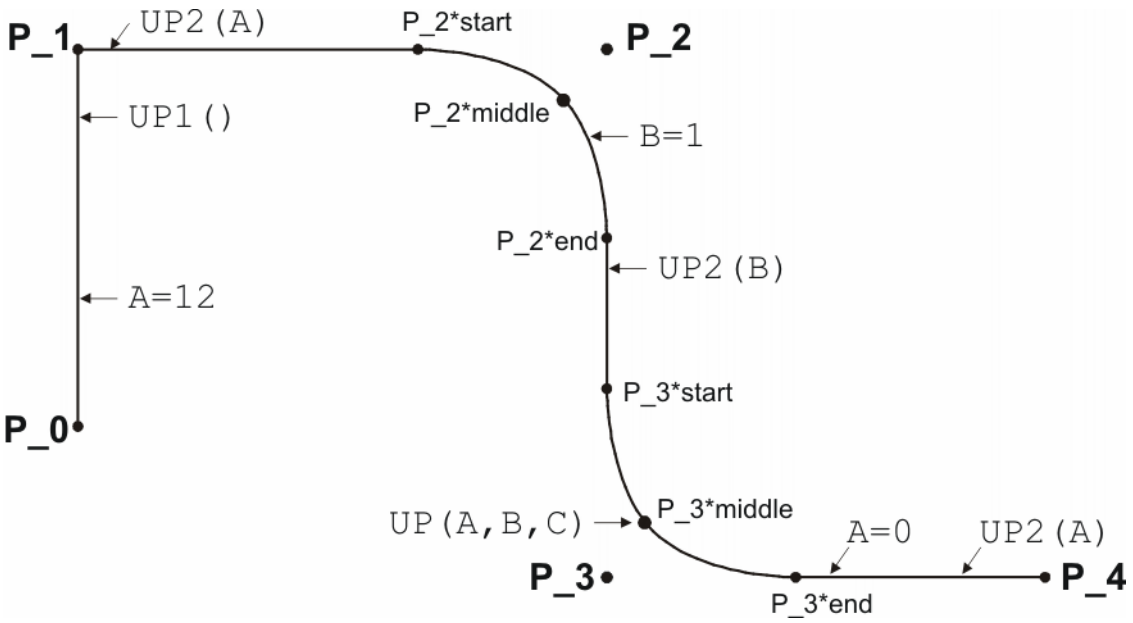


Fig. 11-15: Example of TRIGGER WHEN DISTANCE

11.11.2 TRIGGER WHEN PATH

Description

The Trigger triggers a user-defined statement. The robot controller executes the statement parallel to the robot motion.

The trigger can optionally refer to the start or end point of the motion. The statement can either be triggered directly at the reference point, or it can be shifted in space and/or time.

- The trigger cannot be used for PTP motions.
- If the trigger is used in a spline block, it must not be between the last segment and ENDSPLINE.

Syntax

`TRIGGER WHEN PATH =` *Distance* `<ONSTART> DELAY =` *Time* `DO` *Statement* `<PRIO =` *Priority*`>`

Functions

`PATH` and `DELAY` can call functions. The functions are subject to constraints.

(>>> *11.11.3 "Constraints for functions in the trigger" Page 540*)

Explanation of the syntax

Element	Description
`ONSTART`	Reference point of the trigger • With ONSTART: Start point • Without ONSTART: End point (>>> *11.11.2.1 "Reference point for approximate positioning – overview" Page 537*)
Distance	Type: REAL; variable, constant or function; unit: mm (except in the case of PTP splines without unit) Shift in space relative to the reference point. If no shift in space is desired, set *Distance* = 0. • Negative value: Offset towards the start of the motion • Positive value: Offset towards the end of the motion (>>> *"Max. offset" Page 534*)
Time	Type: REAL; variable, constant or function; unit: ms Shift in time relative to *Distance*. If no shift in time is desired, set *Time* = 0. • Negative value: Offset towards the start of the motion • Positive value: Trigger is switched after *Time* has elapsed. (>>> *"Max. offset" Page 534*)
Statement	Possible: • Assignment of a value to a variable **Note:** There must be no runtime variable on the left-hand side of the assignment. • OUT statement; PULSE statement; CYCFLAG statement • Subprogram call. In this case, *Priority* must be specified.
Priority	Type: INT; variable or constant Priority of the trigger. Only relevant if *Statement* calls a subprogram, and then obligatory. Priorities 1, 2, 4 to 39 and 81 to 128 are available. Priorities 40 to 80 are reserved for cases in which the priority is automatically assigned by the system. If the priority is to be assigned automatically by the system, the following is programmed: `PRIO = -1`. If several triggers call subprograms at the same time, the trigger with the highest priority is processed first, then the triggers of lower priority. 1 = highest priority.

If a trigger calls a subprogram, it counts as an active interrupt until the subprogram has been executed. Up to 32 interrupts may be active at any one time.

Max. offset

The switching point can only be offset within certain limits. If larger, and thus invalid, offsets are programmed, the robot controller switches the trig-

ger at the permissible limit at the latest. In T1/T2, it generates a corresponding message.

Maximum offset for *Distance* + negative *Time* value:

The limits apply to the entire offset, comprising shift in space and negative shift in time.

Maximum negative offset ...	Maximum positive offset ...
Up to start point (provided that this is not approximated)	Up to end point (provided that this is not approximated)
If the start point is an approximate positioning point: • If the start point is an approximated PTP point: Up to the end of the approximate positioning arc • If the start point is a different approximated point: Up to the start of the approximate positioning arc	If the end point is an approximate positioning point: • In the case of homogenous approximate positioning: up to the next exact positioning point after the TRIGGER statement • In the case of mixed approximate positioning (spline): up to the switching point an ON-START trigger with PATH = 0 would have if it were in the motion to which approximate positioning is being carried out. (>>> *11.11.2.3 "Reference point for mixed approximate positioning (spline)" Page 539*) • In the case of mixed approximate positioning (LIN/CIRC/PTP): up to the start of the approximate positioning arc

Maximum offset for positive *Time* value:

The maximum positive shift in time is 1,000 ms. Any shift in time between 0 and 1,000 ms will be switched, even if the program has already been deselected in the meantime!

Example

```
LIN P_2 C_DIS
TRIGGER WHEN PATH = -20.0 DELAY= -10 DO $OUT[2]=TRUE
LIN P_3 C_DIS
LIN P_4 C_DIS
LIN P_5
```

In the diagram, the approximate position in which the $OUT[2]=TRUE statement would be triggered is indicated by an arrow.

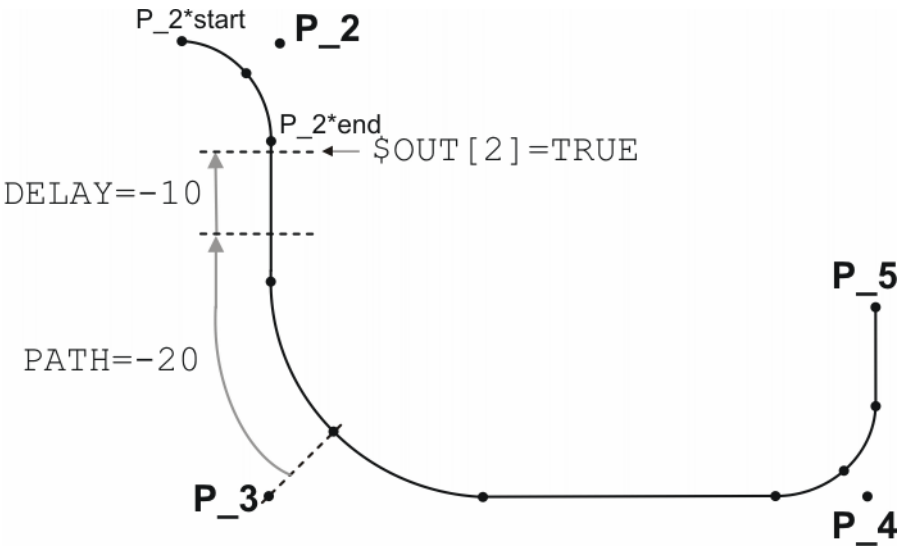


Fig. 11-16: Example of TRIGGER WHEN PATH

Switching range: P_2*start to P_5.

If P_2 were not approximated, the switching range would be P_2 to P_5.

The switching range goes to P_5 because P_5 is the next exact positioning point after the TRIGGER statement. If P_3 were not approximated, the switching range would be P_2 to P_3, as P_3 is the next exact positioning point in the program after the Trigger statement.

Example

```
1  PTP P0
2  SPLINE
3  SPL P1
4  SPL P2
5  SPL P3
6  SPL P4
7  TRIGGER WHEN PATH=0 ONSTART DELAY=10 DO $OUT[5]=TRUE
8  SCIRC P5, P6
9  SPL P7
10 TRIGGER WHEN PATH=-20.0 DELAY=0 DO SUBPR_2() PRIO=-1
11 SLIN P8
12 ENDSPLINE
```

The Trigger in line 10 would have the same result if it was positioned directly before the spline block (i.e. between line 1 and line 2). In both cases, it refers to the last point of the spline motion: P8.

It is advisable, however, to position the trigger as shown in the example, and not directly before the spline block.

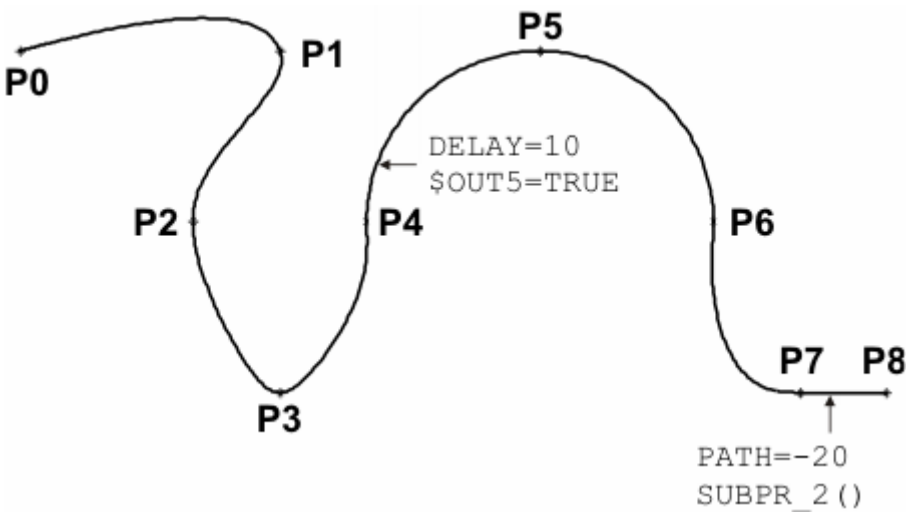


Fig. 11-17: Example of TRIGGER WHEN PATH (for spline)

11.11.2.1 Reference point for approximate positioning – overview

Where is the reference point of a PATH trigger if the start or end point is approximated?

This depends primarily on whether homogenous or mixed approximate positioning is being carried out.

Homogenous

Homogenous approximate positioning

- From a CP spline motion to a CP spline motion
- From a PTP spline motion to a PTP spline motion
- From a LIN or CIRC spline motion to a LIN or CIRC spline motion

Every spline motion can be a spline block or an individual instruction.

(>>> *11.11.2.2 "Reference point for homogenous approximate positioning" Page 537*)

Mixed

Mixed approximate positioning

In this case, the position of the reference point also depends on whether the motions are spline motions or conventional motions.

- From a CP spline motion to a PTP spline motion or vice versa

  Every spline motion can be a spline block or an individual instruction.

  (>>> *11.11.2.3 "Reference point for mixed approximate positioning (spline)" Page 539*)
- From a PTP spline motion to a LIN or CIRC spline motion or vice versa

  (>>> *11.11.2.4 "Reference point for mixed approximate positioning (LIN/CIRC/PTP)" Page 540*)

11.11.2.2 Reference point for homogenous approximate positioning

The principle is explained here using an example with CP spline blocks. It also applies to other types of homogenous approximate positioning.

Example

```
SPLINE
...
SLIN P2
TRIGGER WHEN PATH=0 DELAY=0 DO ... ;Trigger 1
SLIN P3
ENDSPLINE C_SPL
SPLINE
TRIGGER WHEN PATH=0 ONSTART DELAY=0 DO ... ;Trigger 2
SLIN P4
...
ENDSPLINE
```

Triggers 1 and 2 both refer to P3. P3 is approximated. The robot controller transfers the point onto the approximate positioning arc by a distance corresponding to the approximation distance (= P3').

End point approximated

Reference point of Trigger 1:

The robot controller calculates how far the distance would be from the start of the approximate positioning arc to the end point with exact positioning. This distance is then applied to the approximate positioning arc.

The distance $\mathbf{P_{StartApprox}} \rightarrow \mathbf{P3}$ is the same as $\mathbf{P_{StartApprox}} \rightarrow \mathbf{P3'_{Trigger\ 1}}$.

Fig. 11-18: Trigger 1: Reference point for homogenous approximate positioning

$\mathbf{P_{StartApprox}}$	Start of the approximate positioning arc
P3	Reference point for exact positioning
$\mathbf{P3'_{Trigger\ 1}}$	Reference point for approximate positioning

Start point approximated

Reference point of Trigger 2:

The robot controller calculates how far the distance would be from the end of the approximate positioning arc back to the start point with exact positioning. This distance is then applied to the approximate positioning arc.

The distance $\mathbf{P_{EndApprox}} \rightarrow \mathbf{P3}$ is the same as $\mathbf{P_{EndApprox}} \rightarrow \mathbf{P3'_{Trigger\ 2}}$.

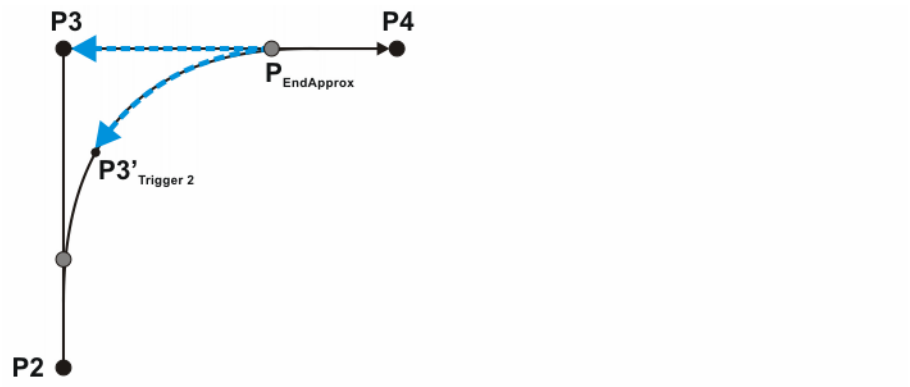


Fig. 11-19: Trigger 2: Reference point for homogenous approximate positioning

$P_{EndApprox}$	End of the approximate positioning arc
P3	Reference point for exact positioning
$P3'_{Trigger\ 2}$	Reference point for approximate positioning

11.11.2.3 Reference point for mixed approximate positioning (spline)

Example

```
PTP_SPLINE
...
SPTP P2
TRIGGER WHEN PATH=0 DELAY=0 DO ... ;Trigger 1
SPTP P3
ENDSPLINE C_SPL

SPLINE
TRIGGER WHEN PATH=0 ONSTART DELAY=0 DO ... ;Trigger 2
SLIN P4
...
ENDSPLINE
```

Triggers 1 and 2 both refer to P3. P3 is approximated.

Start point approximated

Reference point of Trigger 2:

This reference point must be considered first, as the reference point of Trigger 1 refers to it!

The reference point is determined in the same way as for homogenous approximate positioning.

(>>> *"Start point approximated" Page 538*)

End point approximated

Reference point of Trigger 1:

The reference point for Trigger 1 is in the same position as that for Trigger 2.

The distance $P_{StartApprox}$ → $P3'_{Trigger\ 1}$ is generally shorter than $P_{StartApprox}$ → **P3**.

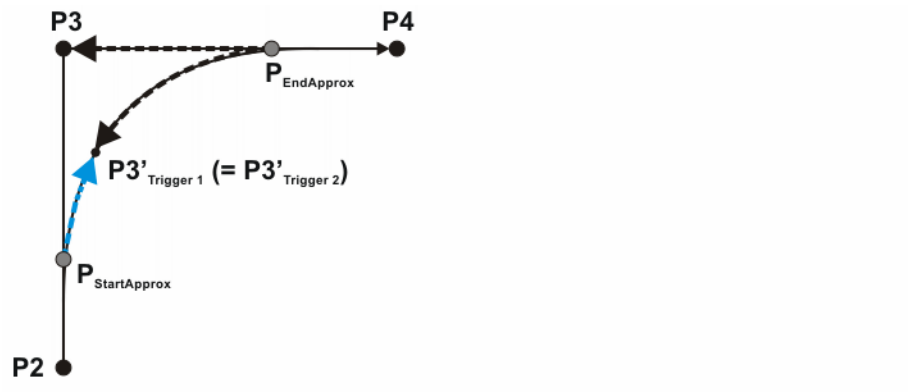


Fig. 11-20: Trigger 1: Reference point for mixed approximate positioning

$P_{StartApprox}$	Start of the approximate positioning arc
$P_{EndApprox}$	End of the approximate positioning arc
P3	Reference point for exact positioning
P3'	Reference point for approximate positioning

If Trigger 1 were to be shifted to between **$P_{StartApprox}$** and **P3'**, the exact position would be determined as follows:

The robot controller calculates the percentage of the distance **$P_{StartApprox}$** → **P3** at which the switching point would be located if the end point were an exact positioning point. This proportion is then applied to the approximate positioning arc. The switching point is thus at x% of the distance **$P_{StartApprox}$** → **$P3'_{Trigger\ 1}$**

11.11.2.4 Reference point for mixed approximate positioning (LIN/CIRC/PTP)

Start point approximated

PTP-CP approximate positioning:

The reference point is at the end of the approximate positioning arc.

End point approximated

CP-PTP approximate positioning:

The reference point is at the start of the approximate positioning arc.

11.11.3 Constraints for functions in the trigger

The values for `DELAY` and `PATH` can be assigned using functions. The following constraints apply to these functions:

- The KRL program containing the function must have the attribute **Hidden**. (>>> *7.4.2 "Displaying or modifying properties of files and folders" Page 293*)
- The function must be globally valid.
- The functions may only contain the following statements or elements:
  - Value assignments
  - IF statements
  - Comments
  - Blank lines

- RETURN
- Read system variable
- Call predefined KRL function

11.11.4 Useful system variables for working with PATH triggers

11.11.4.1 $DIST_NEXT

Description

$DIST_NEXT specifies the length of the path from the current TCP position to the next taught point.

Type: REAL. Unit:

- For CP motions (spline and conventional): mm
- For SPTP motions: No unit

$DIST_NEXT cannot be used for PTP motions. The value is always zero in this case.

$DIST_NEXT is write-protected.

Procedure

$DIST_NEXT can be used as an aid for programming PATH triggers without ONSTART. It can be used to determine the value that must be assigned to the PATH parameter.

1. Move to the position on the path where the switching point is to be located.
2. Read the system variable.
3. Program the trigger before the next point.
   - Program the trigger without ONSTART.
   - Assign the value of the system variable to the PATH parameter.

11.11.4.2 $DIST_LAST

Description

$DIST_LAST specifies the length of the path from the current TCP position to the previous taught point. The value is generally positive.

Type: REAL. Unit:

- For CP motions (spline and conventional): mm
- For SPTP motions: No unit

$DIST_LAST cannot be used for PTP motions. The value is always zero in this case.

$DIST_LAST is write-protected.

Procedure

$DIST_LAST can be used as an aid for programming PATH triggers with ONSTART. It can be used to determine the value that must be assigned to the PATH parameter.

1. Move to the position on the path where the switching point is to be located.
2. Read the system variable.
3. Program the trigger after the previous point.

- Program the trigger with ONSTART.
- Assign the value of the system variable to the PATH parameter.

11.12 Communication

Information about the following statements is contained in the Expert documentation CREAD/CWRITE.

- CAST_FROM
- CAST_TO
- CCLOSE
- CHANNEL
- CIOCTL
- COPEN
- CREAD
- CWRITE
- SREAD
- SWRITE

11.13 Operators

In each operation, the compiler checks the legitimacy of the operands.

11.13.1 Arithmetic operators

Description

All 4 basic arithmetic operations are permissible in KRL.

Operator	Description
+	Addition or positive sign
-	Subtraction or negative sign
*	Multiplication
/	Division

The arithmetic operators can be applied to the data types INT and REAL.

Operand	Operand	Result
INT	INT	INT
INT	REAL	REAL
REAL	REAL	REAL

If the result of an INT division is not an integer, it is cut off at the decimal point.

Examples

```
DEF ARITH()
DECL INT A,B,C,D,E
DECL REAL K,L,M
INI
A = 2 ;A=2
B = 9.8 ;B=10
C = 9.50 ;C=10
```

```
D = 9.48 ;D=9
E = 7/4 ;E=1
K = 3.5 ;K=3.5
L = 1.0 ;L=1.0
M = 3 ;M=3.0
...
A = A * E ;A=2
B = B - 'HB' ;B=-1
E = E + K ;E=5
K = K * 10 ;K=35.0
L = 10/4 ;L=2.0
L = 10/4.0 ;L=2.5
L = 10/4. ;L=2.5
L = 10./4 ;L=2.5
E = 10./4. ;E=3
M = (10/3) * M ;M=9.0
END
```

11.13.2 Geometric operator

Description

Positions can be geometrically added using the geometric operator. The geometric addition is also called a “frame operation”.

The geometric operator is symbolized by a colon “:” in KRL.

The geometric operator is suitable, for example, for the following purposes:

- Shifting positions to adapt them to a modified workpiece size
- Return motion strategies

Example

This statement causes the tool to retract 100 mm against the tool direction, irrespective of the position at which the robot is currently located.

```
LIN $POS_ACT : {x -100, y 0, z 0, a 0, b 0, c 0}
```

The precondition is that the tool direction is located along the X axis.

$POS_ACT is a system variable of structure type E6POS and contains the current Cartesian robot position.

Linked types

The geometric operator can link the data types FRAME and POS/E6POS.

The components X, Y, Z, A, B and C must be assigned a value. The components S and T remain unaffected by the operation and therefore do not have to be assigned a value.

The result always has the data type of the operand on the far right.

Operation with 2 operands:

Left	:	Right	Result
POS	:	POS	POS
POS	:	FRAME	FRAME
FRAME	:	FRAME	FRAME
FRAME	:	POS	POS

Examples of operations with 3 operands:

Left	:	Middle	:	Right	Result
POS	:	POS	:	POS	POS
POS	:	POS	:	FRAME	FRAME
POS	:	FRAME	:	FRAME	FRAME
FRAME	:	FRAME	:	POS	POS

Meaning of the operands

How can one visualize what the operands mean?

This is illustrated using the previous example for a return motion:

Left operand	:	Right operand
`$POS_ACT`	:	`{x -100, y0, z0, a0, b0, c0}`
		Go to this destination …
… relative to the coordinates and orientation of this position.		

11.13.2.1 Sequence of the operands

The result of a geometric addition differs according to the sequence of the operands. The following example illustrates this graphically.

- A = {x 1, y 1, z 0, a 0, b 0, c 0}
- B = {x 3, y 2, z 0, a -45, b 0, c 0}
- CS = original coordinate system

The result of an operation can be calculated using KRL. It specifies the position of the right-hand operand relative to the coordinate system of the left-hand operand.

Sequence A:B

R = A : B means:

- A refers to CS.
- B refers to A.

The result specifies the position of B relative to CS:

R = {x 4, y 3, a -45}

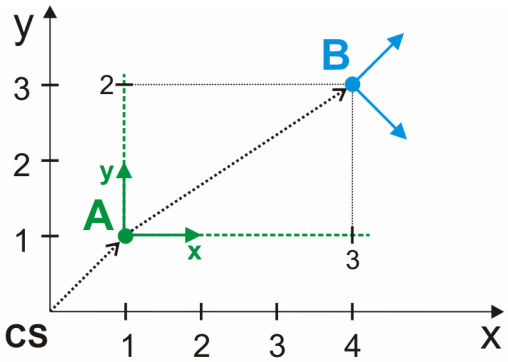


Fig. 11-21: R = A : B

Sequence B:A

R = B : A means:

- B refers to CS.

- A refers to B.

The result specifies the position of A relative to CS:
R = {x 4.414, y 2, a -45}

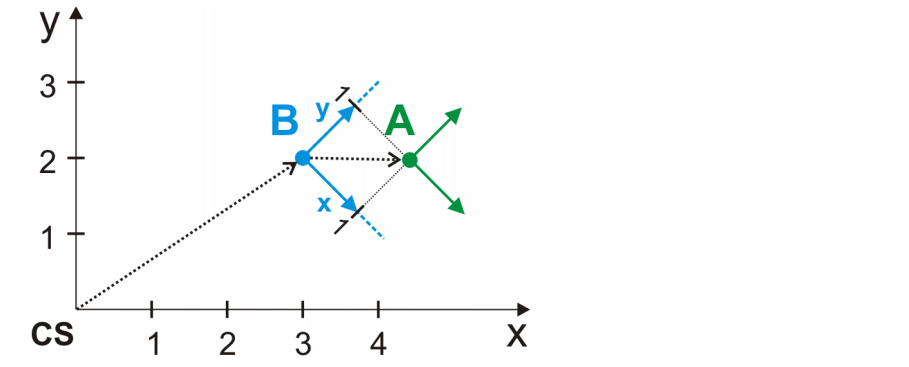


Fig. 11-22: R = B : A

11.13.2.2 Example of a double operation

Description

This example shows how multiple coordinate systems can be linked.

In order to show the effect of the operations, the robot is moved to the origin of each coordinate system and of the operation. The robot waits there for 2 seconds to highlight the position. In order to illustrate the change in orientation, the tip of the tool first moves 100 mm in the X direction, then 100 mm in the Y direction and finally 100 mm in the Z direction.

Program

```
DEF geo_operator( )
DECL AXIS home
DECL FRAME ref_pos_x, ref_pos_y, ref_pos_z
DECL FRAME My_BASE[2]
...
INI
home={AXIS: A1 0,A2 -90,A3 90,A4 0,A5 30,A6 0}
$BASE={X 1000,Y 0,Z 1000,A 0,B 0,C 0}
ref_pos_X={X 100,Y 0,Z 0,A 0,B 0,C 0}
ref_pos_Y={X 100,Y 100,Z 0,A 0,B 0,C 0}
ref_pos_Z={X 100,Y 100,Z 100,A 0,B 0,C 0}
My_BASE[1]={X 200,Y 100,Z 0,A 0,B 0,C 180}
My_BASE[2]={X 0,Y 200,Z 250,A 0,B 90,C 0}
...
PTP home
PTP {X 0,Y 0,Z 0,A 0,B 0,C 0}
WAIT SEC 2
PTP ref_pos_X
PTP ref_pos_Y
PTP ref_pos_Z
PTP My_BASE[1]
WAIT SEC 2
PTP My_BASE[1]:ref_pos_X
PTP My_BASE[1]:ref_pos_Y
PTP My_BASE[1]:ref_pos_Z
PTP My_BASE[1]:My_BASE[2]
WAIT SEC 2
```

```
26 PTP My_BASE[1]:My_BASE[2]:ref_pos_X
27 PTP My_BASE[1]:My_BASE[2]:ref_pos_Y
28 PTP My_BASE[1]:My_BASE[2]:ref_pos_Z
29 PTP My_BASE[2]:My_BASE[1]
30 WAIT SEC 2
31 PTP My_BASE[2]:My_BASE[1]:ref_pos_X
32 PTP My_BASE[2]:My_BASE[1]:ref_pos_Y
33 PTP My_BASE[2]:My_BASE[1]:ref_pos_Z
34 PTP home
35 END
```

Line	Description
8 … 10	Initialization of 3 frames for the motion in the X, Y and Z directions.
11, 12	Initialization of 2 user-specific coordinate systems. These serve as examples for the operations.
14	Move to the origin of the $BASE coordinate system.
16 … 18	In $BASE, first move 100 mm in the X direction, then 100 mm in the Y direction and finally 100 mm in the Z direction.
19	In $BASE, move to the origin of the coordinate system My_BASE[1].
21 … 23	Move to the same coordinates as in lines 16 … 18, but this time in the coordinate system My_BASE[1], not in $BASE. i.e. the location of these points in space is different from that in lines 16 … 18.
24	In My_BASE[1], move to the origin of the coordinate system My_BASE[2]. My_BASE[1] itself is located in $BASE.
26 … 28	The robot moves to the same coordinates as in lines 16 … 18, but this time in the coordinate system My_BASE[1]:My_BASE[2].
29	In My_BASE[2], move to the origin of the coordinate system My_BASE[1]. My_BASE[2] itself is located in $BASE.
31 … 33	The robot moves to the same coordinates as in lines 16 … 18, but this time in the coordinate system My_BASE[2]:My_BASE[1].

11.13.3 Relational operators

Description

Using relational operators, it is possible to form logical expressions. The result of a comparison is always of type BOOL.

Operator	Description	Permissible data types
==	equal to	INT, REAL, CHAR, ENUM, BOOL
<>	not equal to	

Operator	Description	Permissible data types
>	greater than	INT, REAL, CHAR, ENUM
<	less than	
>=	greater than or equal to	
<=	less than or equal to	

- Operand combinations of INT, REAL and CHAR are permissible.

  The comparison of numeric values (INT, REAL) and character values (CHAR) is permissible because each ASCII character is assigned an ASCII code. The code is a number.
- A BOOL type may only be compared with a BOOL type.
- An ENUM type may only be compared with the same ENUM type.

In the case of REAL values, the test for equality or inequality is of only limited use: due to the limited number of places after the floating point, rounding errors are possible. These can result in identical formulae having different values.

Examples

Multiple comparisons are also permissible:

```
...
DECL BOOL A, B
...
B= 10 < 3 ;B=FALSE
A = 10/3 == 3 ;A=TRUE
B = ((B == A) <> (10.00001 >= 10)) == TRUE ;B=TRUE
A = "F" < "Z" ;A=TRUE
...
```

Example of a comparison with an ENUM type:

```
DEF TEST()
ENUM color_typ orange, blue
DECL BOOL A
DECL color_typ KUKA_color, my_color
INI
KUKA_color = #orange
my_color = #orange
...
A = my_color == KUKA_color ;A=TRUE
END
```

11.13.4 Logic operators

Description

Logic operators are used for performing logic operations on Boolean variables, constants and simple logic expressions, as are formed with the aid of relational operators.

Operator	Number of operands	Description
NOT	1	Inversion
AND	2	Logic AND

Operator	Number of operands	Description
OR	2	Logic OR
EXOR	2	Exclusive OR

The operands of a logic operation must be of type BOOL. The result is also always of type BOOL.

The following table shows the results of the possible operations:

Operation		NOT A	A AND B	A OR B	A EXOR B
A = TRUE	B = TRUE	FALSE	TRUE	TRUE	FALSE
A = TRUE	B = FALSE	FALSE	FALSE	TRUE	TRUE
A = FALSE	B = TRUE	TRUE	FALSE	TRUE	TRUE
A = FALSE	B = FALSE	TRUE	FALSE	FALSE	FALSE

The table also applies to operations with bit operators.

Examples

Multiple operations are also permissible.

```
...
DECL BOOL A,B,C
...
A = TRUE ;A=TRUE
B = NOT A ;B=FALSE
C = (A AND B)OR NOT (B EXOR NOT A) ;C=TRUE
A = NOT NOT C ;A=TRUE
...
```

11.13.5 Bit operators

Description

Bit operators are used to link whole numbers by performing logic operations on the individual bits of the whole numbers.

The results of the operations correspond to those of the logic operators.

- Bit value 1 corresponds to TRUE.
- Bit value 0 corresponds to FALSE.

Operator	Number of operands	Description
B_NOT	1	Bit-by-bit inversion
B_AND	2	Bit-by-bit ANDing
B_OR	2	Bit-by-bit ORing
B_EXOR	2	Bit-by-bit exclusive ORing

The bit operators can be applied to the data types INT and CHAR.

INT has 32 bits in KRL and has a sign. CHAR has 8 bits and does not have a sign.

In order to be able to follow the results of bit operations, it is necessary to bear in mind that the robot controller interprets signed binary numbers as a twos complement. The most significant bit determines whether the number is positive or negative. For this reason, all bits must be taken into consideration.

In the following examples for B_AND, B_OR and B_EXOR with integer values, the results are positive numbers (most significant bit = 0). The re-

sults can be converted directly into the decimal system in the same way as unsigned values.

The 28 leading zeros of the operands are indicated by “0 0 […]”.

B_AND

	0	0	[…]	0	1	0	1	= 5
	0	0	[…]	1	1	0	0	= 12
B_AND	**0**	**0**	**[…]**	**0**	**1**	**0**	**0**	**= 4**

Fig. 11-23: Example: Linking the integer values 5 and 12

B_OR

	0	0	[…]	0	1	0	1	= 5
	0	0	[…]	1	1	0	0	= 12
B_OR	**0**	**0**	**[…]**	**1**	**1**	**0**	**1**	**= 13**

Fig. 11-24: Example: Linking the integer values 5 and 12

B_EXOR

	0	0	[…]	0	1	0	1	= 5
	0	0	[…]	1	1	0	0	= 12
B_EXOR	**0**	**0**	**[…]**	**1**	**0**	**0**	**1**	**= 9**

Fig. 11-25: Example: Linking the integer values 5 and 12

B_NOT

In this integer example, the operation results in a negative number (most significant bit = 1). The result can thus not be converted to the decimal system in the same way as an unsigned number.

> In order for the user to be able to understand the decimal result of the robot controller, it is necessary to be familiar with the rules for interpreting twos complement numbers. The rules are not dealt with in this documentation.

	0	0	[…]	1	0	1	0	= 10
B_NOT	**1**	**1**	**[…]**	**0**	**1**	**0**	**1**	**= -11**

Fig. 11-26: Example: B_NOT with integer value 10

The decimal result of a B_NOT operation on a signed operand can also be calculated as follows:

1. Decimal value of the operand plus 1
2. Invert sign

Further examples

```
...
DECL INT A
...
A = 10 B_AND 9 ;A=8
A = 10 B_OR 9 ;A=11
A = 10 B_EXOR 9 ;A=3
A = B_NOT 197 ;A=-198
A = B_NOT 'HC5' ;A=-198
A = B_NOT 'B11000101' ;A=-198
A = B_NOT "E" ;A=154
...
```

Setting and checking bits:

B_AND and B_OR can be used to set individual bits of a bit sequence to 1 or 0. The other bits remain unchanged.

- B_AND can be used to set individual bits to 0.
- B_OR can be used to set individual bits to 1.

It is also possible to check whether individual bits are set to 1 or 0.

Example:

A digital output has a bit width of 8 bits. The output can be addressed via the INT variable DIG.

Set bits 1, 2 and 6 to 0:

```
DIG = DIG B_AND 'B10111001'
```

Set bits 0, 2, 3 and 7 to 1:

```
DIG = DIG B_OR 'B10001101'
```

Check whether bits 0 and 7 are set to 1. If so, `my_result` is set to TRUE:

```
DECL BOOL my_result
...
my_result = DIG B_AND ('B10000001') == 'B10000001'
```

Check whether one of the two bits 0 or 7 is set to 1. If so, `my_result` is set to TRUE:

```
DECL BOOL my_result
...
my_result = DIG B_AND ('B10000001') > 0
```

11.13.6 Priority of the operators

The priority specifies the order in which the operators are evaluated within a statement.

Priority	Operator
1	NOT; B_NOT
2	*; /
3	+; -
4	AND; B_AND
5	EXOR; B_EXOR

Priority	Operator
6	OR; B_OR
7	==, <>; <, >, <=, >=

The following general rules apply:

- Bracketed expressions are processed first.
- Non-bracketed expressions are evaluated in accordance with their priority.
- Logic operations with operators of the same priority are evaluated from left to right.

11.14 Mathematical standard functions

Overview

Function	Range of values of argument	Range of values of result
ABS(X) Absolute value	REAL_MIN…REAL_MAX	0 … REAL_MAX
SQRT(X) Square root	0 … REAL_MAX	0 … REAL_MAX
SIN(X) Sine	REAL_MIN…REAL_MAX	-1 … +1
COS(X) Cosine	REAL_MIN…REAL_MAX	-1 … +1
TAN(X) Tangent	REAL_MIN…REAL_MAX	REAL_MIN…REAL_MAX
ACOS(X) Arc cosine	-1 … +1	0 … +180
ATAN2(Y,X) Arc tangent	REAL_MIN…REAL_MAX	-180 … +180

Data type of all functions: REAL. Data type of all arguments: REAL.

Absolute value

ABS(X) calculates the absolute value of X.
Example:

```
B = -3.4
A = 5*ABS(B)                    ;A=17.0
```

Root

SQRT(X) calculates the square root of X.
Example:

```
A = SQRT(16.0801)               ;A=4.01
```

Sine

SIN(X) calculates the sine of angle X.

Example:

```
A = SIN(30)                          ;A=0,5
```

Cosine

COS(X) calculates the cosine of angle X.

Example:

```
B = 2*COS(45)                        ;B=1.41421356
```

Tangent

TAN(X) calculates the tangent of angle X.

Example:

```
C = TAN(45)                          ;C=1.0
```

The tangent of the following absolute values is infinite:

- ±90°
- +90° + k*180° (where k = ±integer)

If an attempt is made to calculate such a value, this leads to an error message.

Arc cosine

ACOS(X) is the inverse function of COS(X).

Example:

```
A = COS(60)                          ;A=0.5
B = ACOS(A)                          ;B=60
```

Arc sine

There is no function predefined for arc sine, the inverse function of SIN(X). However, the arc sine can be calculated very easily on the basis of the relationship SIN(X) = COS(90°-X).

Example:

```
A = SIN(60)                           ;A=0.8660254
B = 90-ACOS(A)                       ;B=60
```

Arc tangent

The tangent of an angle is defined as the opposite side (Y) divided by the adjacent side (X) of a right-angled triangle. If the lengths of the two legs of the triangle are known, it is thus possible to calculate the angle between the adjacent side and the hypotenuse by means of the arc tangent.

In the case of a full circle, the sign of X and Y is of decisive importance. If only the quotient were to be considered, it would only be possible to calculate angles between 0° and 180° by means of the arc tangent. This is normally also the case with pocket calculators: the arc tangent of positive values gives an angle between 0° and 90°. The arc tangent of negative values gives an angle between 90° and 180°.

By specifying X and Y, the quadrant in which the angle is located is unambiguously defined by their signs. Angles in quadrants III and IV can thus also be calculated.

Example:

```
A = ATAN2(0.5,0.5)                ;A=+45
B = ATAN2(0.5,-0.5)               ;B=+135
C = ATAN2(-0.5,-0.5)              ;C=-135
D = ATAN2(-0.5,0.5)               ;D=-45
```

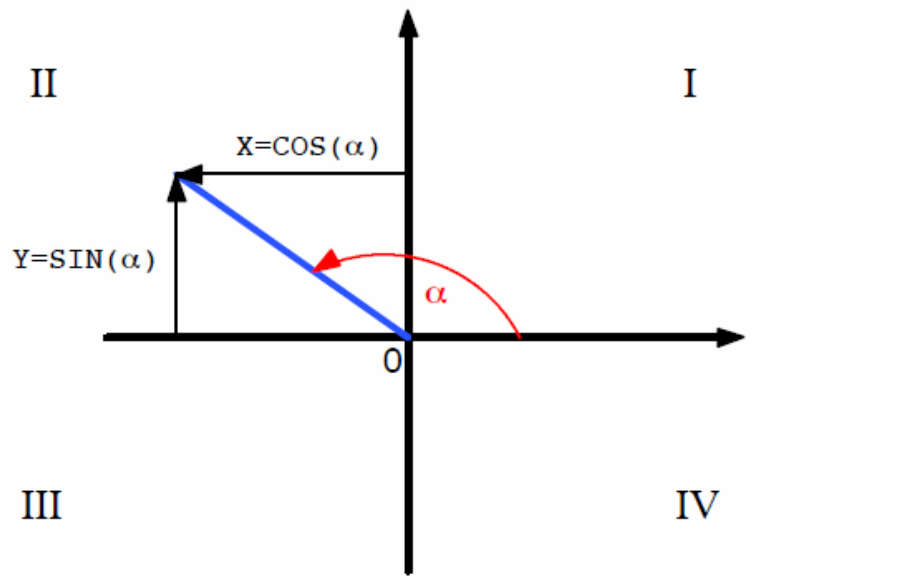


Fig. 11-27: Use of X and Y in the function ATAN(Y,X)

11.15 System functions

11.15.1 DELETE_BACKWARD_BUFFER(): preventing backward start

Description

`DELETE_BACKWARD_BUFFER()` can be used to prevent backward motion for specific motions. The function deletes the recorded forward motions. If the user attempts to perform backward motion, the robot controller generates the following message: *Backward motion not possible: no trace available*.

The function can be used with all interpreters, but only if no main run is active.

The function triggers an advance run stop.

`DELETE_BACKWARD_BUFFER()` refers to backward motion using the Start backwards key. It has no effect on other backward motion functionalities, e.g. backward motion as part of fault strategies in technology packages.

This function cannot be used in the trigger.

11.15.2 FORWARD(): converting axis angles to a Cartesian position

Description

The FORWARD function calculates the Cartesian position in space (BASE system) from the axis angles of the robot and the external axes.

The function can be used for diagnostic purposes.

- External axes are taken into account properly, both in the form of a Robroot kinematic system and as a Base Kinematic system.

- The function can be called by means of the SHOWVAR or SETVAR command, or in a KRL program.

Syntax

result = FORWARD (*axis_values*, *err_status*)

Explanation of the syntax

Element	Description
result	Type: E6POS Variable for the return value Cartesian position relative to the BASE system.
axis_values	Type: E6AXIS Transfer type: IN parameter Axis angles of the robot for which the Cartesian position is to be calculated
err_status	Type: INT Transfer type: OUT parameter Setting whether the transferred axis angles are to be checked against software limit switches: • 0: All axis angles are tested. If they are not within the limits of the software limit switches, *err_status* returns an error code. • <>0: Axis angles are not checked.

Error codes

The variable *err_status* transfers the result of the calculation. If the calculation is not successful, it transfers a value that corresponds to an error code:

Value	Description
-4	The advance run variable $TOOL is invalid.
-3	The advance run variable $BASE is invalid.
-2	*err_status* does not yet have a valid value.
-1	Not all robot axis angles / required external axis angles have been defined.
0	Calculation was successful, no error.
1	Software limit switches violated with the specified axis angles.
2	Errors in the mathematical transformation

11.15.3 INV_POS(): calculating an inverse position

Description

The function INV_POS calculates the inverse position from the robot position of data type FRAME, POS or E6POS. The values for Status, Turn and E1 to E6 remain unchanged.

The function can be used to convert a position to another TOOL and BASE coordinate system.

The function is mainly used in conjunction with the geometric operator (:). (>>> *11.13.2 "Geometric operator" Page 543*)

Syntax

result = `INV_POS(`*position*`)`

Explanation of the syntax

Element	Description
result	Type: E6POS Variable for the return value Inverted position • If the function was called with an argument of type E6POS, Status and Turn remain constant, as do E1 to E6. • If the function was called with an argument of type POS, E1 to E6 are not defined. • If the function was called with an argument of type FRAME, Status and Turn are not defined, nor are E1 to E6.
position	Type: E6POS, POS, FRAME Transfer type: IN parameter Position to be inverted If one or more of the components X, Y, Z, A, B, C is invalid, the robot controller generates an error message.

Example 1

Calculating the current flange position (in the main run) in relation to the WORLD coordinate system $WORLD

```
FlangePos=$BASE_C:$POS_ACT:INV_POS($TOOL_C)
```

Example 2

Converting a point to another coordinate system.

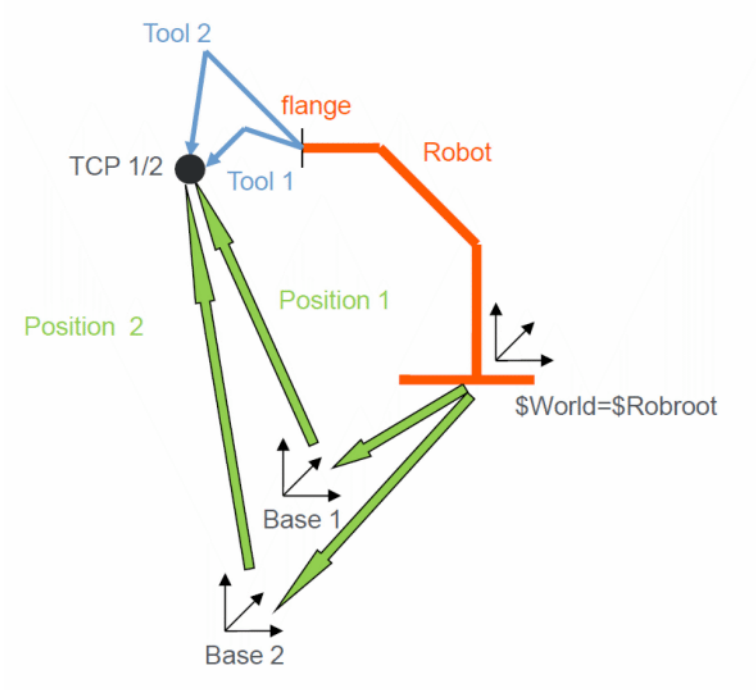


Fig. 11-28: INV_POS example

```
Position2=INV_POS(BASE_DATA[2]):BASE_DATA[1]:Posi-
tion1:INV_POS(TOOL_DATA[1]):TOOL_DATA[2]
```

11.15.4 INVERSE(): converting a Cartesian position to axis angles

Description

The INVERSE function calculates the robot axis angles from a corresponding Cartesian position with external axis angle. It is not strictly necessary to specify the Status and Turn values for the Cartesian position.

The INVERSE function can be used in palletizing, for example, to address the calculated points in PTP. The function can be used to check the validity of the Turn value and adapt it at the end point as required.

- External axes are taken into account properly, both in the form of a Robroot kinematic system and as a Base Kinematic system.
- The function can be called by means of the SHOWVAR or SETVAR command, or in a KRL program.

Syntax

result = INVERSE (*position*, *start_axis*, *err_status*)

Explanation of the syntax

Element	Description
result	Type: E6AXIS Variable for the return value Axis angles at the transferred position
position	Type: E6POS Transfer type: IN parameter Cartesian position (with external axis angles if applicable) relative to the BASE system The robot axis angles for this position are calculated.
start_axis	Type: E6AXIS Transfer type: IN parameter Axis angles of the robot at the start point of the motion
err_status	Type: INT Transfer type: OUT parameter Setting whether the transferred axis angles (*start_axis*) are to be checked against software limit switches: • 0: All axis angles are tested. If they are not within the limits of the software limit switches, *err_status* returns an error code. • <>0: Axis angles are not checked. The calculated axis angles (*result*) are always checked against software limit switches.

Using the start point

The start point *start_axis* is required in the following cases:

- The end point has no status value.

  The system variable `$TARGET_STATUS` defines which status value the end point is to receive:

  – `$TARGET_STATUS=#SOURCE`

    The end point receives the same status as the start point. The status is calculated from the axis angles of the point *start_axis*.

  – `$TARGET_STATUS=#BEST`

    The end point receives the status that enable the robot to cover the shortest possible distance from the start point to the end point in axis space.

- The end point has no turn value.

  For each axis, the permissible turn value is calculated which results in the shortest path between the start point and end point. “Permissible” in this case means within the software limit switches.

- The end point is situated close to a singularity.

  One axis angle must be specified and the one that is dependent on it is calculated. The system variable `$SINGUL_POS[1...3]` is used to set the angle that the end point is to receive:

  – `$SINGUL_POS[1...3]=0`: The angle of the axis is set to 0 degrees.

– `$SINGUL_POS[1...3]=1`: The angle remains the same from the start point to the end point.

Error codes

The variable *err_status* transfers the result of the calculation. If the calculation is not successful, it transfers a value that corresponds to an error code:

Value	Description
-4	The advance run variable $TOOL is invalid.
-3	The advance run variable $BASE is invalid.
-2	*err_status* does not yet have a valid value.
-1	Not all essential components of the *position* variable have been defined. S and T do not strictly need to be defined.
0	Calculation was successful, no error.
1	Software limit switches violated with the specified axis angles.
2	There are no axis angles with which the Cartesian end position can be reached.
3	There are axis angles with which the Cartesian end position can be reached, but the specified Turn would result in a software limit switch being violated.

Example

The KRL program KUE_WEG calculates the status for axis 5 in such a way that the PTP motion of axes 4 and 5 between the start point and end point is as short as possible. It can be integrated into a motion program or used for the offline generation of Cartesian points.

The program is available as standard on every controller:

Directory	C:\KRC\UTIL\KUEWEG
File	kue_weg.src

11.15.5 ROB_STOP(): preventing robot motions

Description

ROB_STOP() and ROB_STOP_RELEASE() can only be used in submit programs.

- ROB_STOP() stops the robot and prevents further motions. This affects all possible motions, irrespective of whether they arise from program execution, manual jogging or command mode.
- ROB_STOP_RELEASE() cancels a blockade caused by ROB_STOP().

If ROB_STOP() is called several times in succession without a ROB_STOP_RELEASE() in between, only the first call has any effect. The subsequent calls have no effect, i.e. they do not trigger a stop or cause a message to be generated.

If ROB_STOP_RELEASE() is called without ROB_STOP() being called first, this has no effect.

Messages

ROB_STOP() triggers the following status message: *Robot stopped by submit*

ROB_STOP_RELEASE() in a Test mode triggers the following acknowledgement message: *Ackn. Robot stopped by submit*

ROB_STOP_RELEASE() in an Automatic mode triggers no message.

Syntax

result = `ROB_STOP` (*stop_type*)

Explanation of the syntax

Element	Description
result	Type: BOOL Variable for the return value. Return value: • TRUE: The stop has been executed. • FALSE: An invalid parameter has been transferred for *stop_type*.
stop_type	Type: ROB_STOP_T Transfer type: IN parameter Stop type to be used to stop the robot: • #RAMP_DOWN: ramp stop • #PATH_MAINTAINING: path-maintaining EMERGENCY STOP Other stop types are not possible.

ProConOS

The “Robot stop” function can also be used from ProConOS. The following functions are available for this:

- PLC_ROB_STOP()
  The desired stop type is defined with PLC_ROB_STOP_RAMP_DOWN or PLC_ROB_STOP_PATH_MAINT.
- PLC_ROB_STOP_RELEASE()

The effect is the same, irrespective of whether a stop is triggered by submit or by ProConOS. ProConOS generates its own message texts; these are:

- *Robot stopped by SoftPLC ({Name of the task calling it})*
- *Ackn. Robot stopped by SoftPLC*

If a stop is requested by both submit and ProConOS, this is indicated in each case by a status message. A maximum of 2 status messages can thus be displayed for an executed stop. In this case, robot motion cannot be resumed until the blockade has been canceled by both submit and ProConOS.

If different stop types are requested by submit and ProConOS, the type actually carried out is generally the one that was requested first.

A stop triggered by ProConOS cannot be canceled by a submit program and vice versa.

11.15.6 SET_BRAKE_DELAY(): reducing the brake delay

Description

The function SET_BRAKE_DELAY can be used to reduce the brake delay with reference to an individual point.

SET_BRAKE_DELAY is intended for use at the end of a cycle. When the robot stops there before the next cycle begins, SET_BRAKE_DELAY can be used to make the brakes close earlier, thereby also causing the drives to be deactivated sooner. Energy can be saved in this way.

Brake delay:

The brake delay is the time after which the brakes are applied when the robot (or the external axis) has reached an exact positioning point. It is irrelevant whether the exact positioning point was programmed as such, or whether it just works out that way because approximate positioning cannot be carried out.

If the robot stops at the point until the time has elapsed, e.g. at the end of the program, the brakes are applied. If the robot resumes motion before the time has elapsed, the brakes are not applied.

The generally applicable brake delay is defined using $BRK_DEL.
$BRK_DEL: Brake delay for robot axes and external axes in command mode (= jogging) and in program mode (default: 20,000 ms)

SET_BRAKE_DELAY can be used to set a lower value for an individual point, i.e. the brakes are applied earlier.

Additional characteristics

SET_BRAKE_DELAY triggers an advance run stop. The advance run stop applies separately for synchronous and asynchronous axes. For example, if a synchronous axis is specified using *axes_nr*, the advance run stop applies for all synchronous axes, but not for any asynchronous axis that may be present.

SET_BRAKE_DELAY can be processed by all interpreters.

SET_BRAKE_DELAY only has an effect if the robot is at an exact positioning point:

- SET_BRAKE_DELAY must come after the point in the program to which it is to apply. Since it triggers an advance run stop, this point is automatically an exact positioning point.
- If it is triggered by a trigger, it can only take effect if the trigger refers to the end point and this is an exact positioning point.

Syntax

result = `SET_BRAKE_DELAY (`*axes_nr*`,` *delay*`)`

Explanation of the syntax

Element	Description
result	Type: INT Variable for the return value. The bits indicate the axes for which *delay* has been set. • Bit n = 0: value was not set for this axis. • Bit n = 1: value was set for this axis. The return value does not indicate whether the brakes were actually applied.
axes_nr	Type: INT Bit array for the axes for which *delay* is to be set. • Bit n = 0: value is not set for this axis. • Bit n = 1: value is set for this axis. The value can be specified in the program as an integer or using bit notation, e.g. “63” or “ 'B111111' ” for “all robot axes”. **Note:** The *delay* value applies to the axes specified in the bit array. If multiple axes are physically present on the same brake channel, however, the *delay* value also applies to the other axes on this channel, provided that there is also a brake closing task available for them. If there is no brake closing task available for the other axes on the same channel, all axes on this channel are not closed.
delay	Type: INT; unit: ms Desired delay. Range of values: • **0** … defined general brake delay If a higher value is defined, it is limited internally to the value of $BRK_DEL. The value 0 is permissible. The actual closing time, however, is always at least as long as the time required mechanically for the brake to close. This is just a few fractions of a second. The exact value depends on the specific axis.

Bit n	11 …	5	4	3	2	1	0
Axis	E6 …	A6	A5	A4	A3	A2	A1

$BRAKE_SIG

The state of the brakes (open or closed) can be displayed by means of the system variable $BRAKE_SIG.

(>>> *"$BRAKE_SIG" Page 264*)

Example

At the end of the following program, the brakes of the robot axes are to be applied as quickly as possible. For this reason, SET_BRAKE_DELAY(63, 0) has been programmed after the last point.

```
1 DEF my_test()
2 DECL INT my_result
```

```
3 DECL INT brake_state
...
4 PTP HOME Vel= 100 % DEFAULT
5 PTP P1 ...
...
6 PTP HOME Vel= 100 % DEFAULT
7 my_result = SET_BRAKE_DELAY(63, 0)
8 brake_state = $BRAKE_SIG
9 END
```

Line	Description
6	Last point in program
7	Here, the brake delay is set to 0 ms for all robot axes for the point in line 6.
8	The monitoring reveals that the brakes are (still) open at this point. The reason for this is that the brakes cannot close in 0 ms; they require a certain time to close for mechanical reasons. Once this time has elapsed, the brakes are closed.

Negative example

It is not generally helpful to use SET_BRAKE_DELAY during a cycle. There are often no points at which the brakes are applied and where this operation would need to be accelerated. On the contrary, the advance run stop triggered by SET_BRAKE_DELAY would actually have a negative effect on the cycle time.

```
1 DEF my_test()
2 DECL INT my_result
...
3 PTP HOME Vel= 100 % DEFAULT
4 PTP P1 C_DIS ...
5 my_result = SET_BRAKE_DELAY(63, 0)
6 ;WAIT SEC 0.5
7 PTP P2 ...
...
```

Line	Description
5	Here, the brake delay is set to 0 ms for all robot axes for P1. P1 is programmed with approximate positioning. Since SET_BRAKE_DELAY triggers an advance run stop, the motion to P1 is carried out with exact positioning. The brakes do not close at P1, however. The reason for this is that the brakes would need the time mechanically required to close. However, on reaching P1, the robot controller immediately starts the next motion. The brakes are thus not applied, even though the delay is set to 0 ms.
6	By contrast, if this line were uncommented, the brakes would close. The robot controller would not immediately start the next motion on reaching P1, but stop at P1 for 0.5 s. This would allow time for the brakes to be applied.

11.15.7 VARSTATE(): polling the state of a variable

Description

VARSTATE() can be used to monitor the state of a variable. VARSTATE() is a function with a return value of type VAR_STATE.

Possible values: #DECLARED, #INITIALIZED, #UNKNOWN

Definitions:

```
ENUM VAR_STATE DECLARED, INITIALIZED, UNKNOWN
```

```
VAR_STATE VARSTATE(CHAR VAR_STR[80]:IN)
```

Example 1

```
 1 DEF PROG1()
 2 INT MYVAR
 3 ...
 4 IF VARSTATE("MYVAR")==#UNKNOWN THEN
 5     $OUT[11]=TRUE
 6 ENDIF
 7 ...
 8 IF VARSTATE("MYVAR")==#DECLARED THEN
 9     $OUT[12]=TRUE
10 ENDIF
11 ...
12 IF VARSTATE("ANYVAR")==#UNKNOWN THEN
13     $OUT[13]=TRUE
14 ENDIF
15 ...
16 MYVAR=9
17 ...
18 IF VARSTATE("MYVAR")==#DECLARED THEN
19     $OUT[14]=TRUE
20 ENDIF
21 ...
22 IF VARSTATE("MYVAR")==#INITIALIZED THEN
23     $OUT[15]=TRUE
24 ENDIF
25 ...
26 END
```

Line	Description
4 … 6	The IF condition is false, as MYVAR has already been declared. Output 11 is not set.
8 … 10	The IF condition is true, as MYVAR has been declared. Output 12 is set.
12 … 14	The IF condition is true, on the condition that there is also no variable with the name ANYVAR in $CONFIG.DAT. Output 13 is set.
18 … 20	The IF condition is false, as MYVAR has not only been declared, but has also already been initialized at this point. Output 14 is not set.
22 … 24	The IF condition is true, as MYVAR has been initialized. Output 15 is set.

Example 2

```
DEF PROG2()
INT MYVAR
INT YOURVAR
DECL VAR_STATE STATUS
...
STATUS=VARSTATE("MYVAR")
UP()
...
STATUS=VARSTATE("YOURVAR")
UP()
...
END
```

```
DEF UP()
...
IF VARSTATE("STATUS")==#DECLARED THEN
    $OUT[100]=TRUE
ENDIF
...
END
```

Explanation of the state polling:

In this example, the state is monitored indirectly, i.e. via an additional variable. The additional variable must be of type VAR_STATE. The keyword DECL must not be omitted in the declaration. The name of the additional variable may be freely selected. In this example it is STATUS.

11.16 Editing string variables

Various functions are available for editing string variables. The functions can be used in SRC files, in SUB files and in the variable correction function.

The functions can be used within IF branches without the return value being explicitly assigned to a variable.

11.16.1 Converting a string variable to a different data type

Description

The functions of type StrTo[…] can be used to convert string variables to a different data type. The following functions are declared in KRL:

- BOOL StrToAXIS (CHAR strValue[256], AXIS value)
- BOOL StrToBOOL (CHAR strValue[256], BOOL value)
- BOOL StrToE3AXIS (CHAR strValue[256], E3AXIS value)
- BOOL StrToE6AXIS (CHAR strValue[256], E6AXIS value)
- BOOL StrToE3POS (CHAR strValue[256], E3POS value)
- BOOL StrToE6POS (CHAR strValue[256], E6POS value)
- BOOL StrToFRAME (CHAR strValue[256], FRAME value)
- BOOL StrToINT (CHAR strValue[256], INT value)
- BOOL StrToPOS (CHAR strValue[256], POS value)
- BOOL StrToREAL (CHAR strValue[256], REAL value)
- BOOL StrToSTRING (CHAR strValue[256], STRING value)

Syntax

StrToAXIS is shown here as an example for the type StrTo[…]:

success = `StrToAXIS(`*string*`,` *value*`)`

Explanation of the syntax

StrToAXIS is explained here as an example for the type StrTo[…]: The other functions are treated analogously.

Element	Description
success	Type: BOOL Variable for the return value • TRUE: Conversion successful • FALSE: Conversion not successful
string	Type: CHAR String variable that is to be converted to a different data type If *string* represents an aggregate, the individual components must be separated by commas.
value	Type: AXIS Variable for the value after conversion

Example 1

```
1 DECL BOOL success, value
2 ...
3 success=StrToBOOL("1", value)
4 success=StrToBOOL("TRUE", value)
```

Line	Description
3	Conversion is not possible, as the string variable “1” does not match the data type BOOL. The return value *success* is FALSE.
4	Conversion is possible. The return value *success* is TRUE. The value after conversion is TRUE.

Example 2

```
1 DECL FRAME value
2 DECL BOOL success
3 ...
4 success=StrToFRAME("{X 10, Y 50, C 45}",value)
```

Line	Description
4	Conversion is possible. The return value *success* is TRUE. The value after conversion is “X 10, Y 50, C 45”. Missing frame components are not initialized and thus remain without a value. If there are already components present in the target variable, they are deleted.

Example 3

```
1 DECL AXIS value
2 DECL BOOL success
3 ...
4 success=StrToAXIS("{A1 45}",value)
5 success=StrToAXIS("{E1 100}",value)
6 success=StrToAXIS("{A1 90, E1 100}",value)
```

Line	Description
4	Conversion is possible, as A1 is a component of AXIS. The return value *success* is TRUE.
5	Conversion is not possible, as E1 is not a component of AXIS. The return value *success* is FALSE.
6	Conversion is possible. The value of E1 is lost. The return value *success* is TRUE.

11.16.2 String variable length in the declaration

Description

The function `StrDeclLen()` determines the length of a string variable according to its declaration in the declaration section of a program.

Syntax

Length = `StrDeclLen(`*StrVar[]*`)`

Explanation of the syntax

Element	Description
Length	Type: INT Variable for the return value. Return value: Length of the string variable as declared in the declaration section
StrVar[]	Type: CHAR array String variable whose length is to be determined Since the string variable StrVar[ ] is an array of type CHAR, individual characters and constants are not permissible for length determination.

Example

```
1 CHAR ProName[24]
2 INT StrLength
…
3 StrLength = StrDeclLen(ProName)
4 StrLength = StrDeclLen($Trace.Name[ ])
```

Line	Description
3	StrLength = 24
4	StrLength = 64

11.16.3 String variable length after initialization

Description

The function `StrLen()` determines the length of the character string of a string variable as defined in the initialization section of the program.

Syntax

Length = `StrLen(`*StrVar*`)`

Explanation of the syntax

Element	Description
Length	Type: INT Variable for the return value. Return value: Number of characters currently assigned to the string variable
StrVar	Type: CHAR Character string or variable whose length is to be determined

Example

```
1 CHAR PartA[50]
2 INT AB
…
3 PartA[] = "This is an example"
4 AB = StrLen(PartA[])
```

Line	Description
4	AB = 18

11.16.4 Deleting the contents of a string variable

Description

The function `StrClear()` deletes the contents of a string variable.

Syntax

Result = `StrClear(`*StrVar[]*`)`

Explanation of the syntax

Element	Description
Result	Type: BOOL Variable for the return value. Return value: • The contents of the string variable have been deleted: TRUE • The contents of the string variable have not been deleted: FALSE
StrVar[]	Type: CHAR array Variable whose character string is to be deleted

Example

```
IF (NOT StrClear($Loop_Msg[])) THEN
HALT
ENDIF
```

The function can be used within IF branches without the return value being explicitly assigned to a variable. This applies to all functions for editing string variables.

11.16.5 Extending a string variable

Description

The function `StrAdd()` can be used to expand a string variable with the contents of another string variable.

Syntax

Sum = `StrAdd(`*StrDest[], StrToAdd[]*`)`

Explanation of the syntax

Element	Description
Sum	Type: INT Variable for the return value. Return value: Sum of StrDest[ ] and StrToAdd[ ] If the sum is longer than the previously defined length of *StrDest[ ]*, the return value is 0. This is also the case if the sum is greater than 470 characters.
StrDest[]	Type: CHAR array The string variable to be extended Since the string variable StrDest[ ] is an array of type CHAR, individual characters and constants are not permissible.
StrToAdd[]	Type: CHAR array The character string by which the variable is to be extended

Example

```
1 DECL CHAR A[50], B[50]
2 INT AB, AC
…
3 A[] = "This is an "
4 B[] = "example"
5 AB = StrAdd(A[],B[])
```

Line	Description
5	A[ ] = “This is an example” AB = 18

11.16.6 Searching a string variable

Description

The function `StrFind()` can be used to search a string variable for a character string.

Syntax

Result = `StrFind(`*StartAt, StrVar[], StrFind[], CaseSens*`)`

Explanation of the syntax

Element	Description
Result	Type: INT Variable for the return value. Return value: Position of the first character found. If no character is found, the return value is 0.
StartAt	Type: INT The search is started from this position.
StrVar[]	Type: CHAR array The string variable to be searched
StrFind[]	Type: CHAR array The character string that is being looked for.
CaseSens	• #CASE_SENS: Upper and lower case are taken into consideration. • #NOT_CASE_SENS: Upper and lower case are ignored.

Example

```
1 DECL CHAR A[5]
2 INT B
3 A[]="ABCDE"
4 B = StrFind(1, A[], "AC", #CASE_SENS)
5 B = StrFind(1, A[], "a", #NOT_CASE_SENS)
6 B = StrFind(1, A[], "BC", #Case_Sens)
7 B = StrFind(1, A[], "bc", #NOT_CASE_SENS)
```

Line	Description
4	B = 0
5	B = 1
6	B = 2
7	B = 2

11.16.7 Comparing the contents of string variables

Description

The function `StrComp()` can be used to compare two string variables.

Syntax

Comp = `StrComp(`*StrComp1[], StrComp2[], CaseSens*`)`

Explanation of the syntax

Element	Description
Comp	Type: BOOL Variable for the return value. Return value: • The character strings match: TRUE • The character strings do not match: FALSE
StrComp1[]	Type: CHAR array String variable that is compared with StrComp2[].
StrComp2[]	Type: CHAR array String variable that is compared with StrComp1[].
CaseSens	• #CASE_SENS: Upper and lower case are taken into consideration. • #NOT_CASE_SENS: Upper and lower case are ignored.

Example

```
1 DECL CHAR A[5]
2 BOOL B
3 A[]="ABCDE"
4 B = StrComp(A[], "ABCDE", #CASE_SENS)
5 B = StrComp(A[], "abcde", #NOT_CASE_SENS)
6 B = StrComp(A[], "abcd", #NOT_CASE_SENS)
7 B = StrComp(A[], "acbde", #NOT_CASE_SENS)
```

Line	Description
4	B = TRUE
5	B = TRUE
6	B = FALSE
7	B = FALSE

11.16.8 Copying a string variable

Description

The function `StrCopy()` can be used to copy the contents of a string variable to another string variable.

Syntax

Copy = `StrCopy(`*StrDest[], StrSource[]*`)`

Explanation of the syntax

Element	Description
Copy	Type: BOOL Variable for the return value. Return value: • The string variable was copied successfully: TRUE • The string variable was not copied: FALSE
StrDest[]	Type: CHAR array The character string is copied to this string variable. Since StrDest[ ] is an array of type CHAR, individual characters and constants are not permissible.
StrSource[]	Type: CHAR array The contents of this string variable are copied.

Example

```
1 DECL CHAR A[25], B[25]
2 DECL BOOL C
3 A[] = ""
4 B[] = "Example"
5 C = StrCopy(A[], B[])
```

Line	Description
5	A[ ] = “Example” C = TRUE

12 Submit interpreter

12.1 Function of the submit interpreters

Submit interpreter

Motion programs run in the robot interpreter. Alongside the robot interpreter, the following additional interpreters also exist on the robot controller:

- One system submit interpreter

  The program SPS.SUB runs in the system submit interpreter.
- 7 extended submit interpreters

  Users can create their own SUB programs and assign these to the extended submits.

Use

SUB programs can perform operator control or monitoring tasks. Examples: monitoring of safety equipment; monitoring of a cooling circuit. This means that no PLC is required for smaller applications, as the robot controller can perform such tasks by itself.

- All the submit interpreters can be used simultaneously.
- The extended submits are intended for user-specific submit tasks.
- The system submit and the sps.sub program are intended for controller-internal submit tasks.

By default, the program SPS.SUB is assigned to the system submit interpreter. It is strongly recommended not to change this assignment. It is necessary to enable the controller-internal submit tasks to be executed. If this assignment is changed, this can affect the functionality of technology packages and other options.

WARNING

Submit interpreters must not be used for time-critical applications! A PLC must be used in such cases. Reasons:

- The submit interpreters share system resources with the robot interpreter, which has the higher priority. Submit interpreters are thus not executed at the robot controller's interpolation cycle rate of 12 ms. Furthermore, the runtime of the submit interpreters is irregular.
- The runtime of the submit interpreters is influenced by the number of lines in the SUB program. Even comment lines and blank lines have an effect.

If a system file, e.g. $config.dat or $custom.dat, is modified in such a way that errors are introduced, the currently active submit interpreters are automatically deselected. Once the error in the system file has been rectified, the submit interpreters must be reselected manually.

States

A submit interpreter can have the following states:

State	Description
ACTIVE	A program is selected and running in the submit interpreter.
STOP	A program is selected in the submit interpreter, but has been stopped.

State	Description
RESET	A program is selected in the submit interpreter, but has been stopped and reset.
FREE	No program is selected in the submit interpreter. FREE can have the following variants: • FREE_PRESELECTED: A program is assigned to the submit interpreter in the **Submit interpreter** window under **Available modules**. The program is not selected. • FREE_UNSELECTED: No program is assigned to the submit interpreter in the **Submit interpreter** window under **Available modules**.

12.2 Status indicator for the submit interpreter

The status bar of the KUKA smartHMI contains a group indicator of the state of the submit interpreters. The state shown is always the highest state currently applying among all the submit interpreters.

The sequence of states (in descending order) is:

- ACTIVE, STOP, RESET, FREE

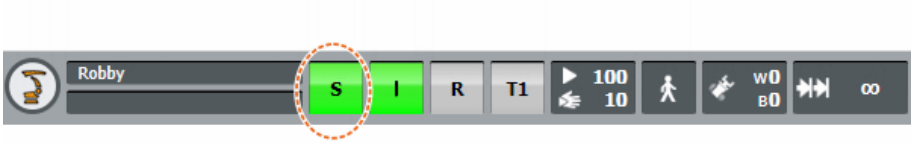


Fig. 12-1: Submit interpreter status indicator

Icon	Color / description
	Green At least one submit interpreter is in the ACTIVE state.
	Red At least one submit interpreter is in the STOP state. No submit interpreter is in the ACTIVE state.
	Yellow At least one submit interpreter is in the RESET state. No submit interpreter is in the STOP state. No submit interpreter is in the ACTIVE state.
	Gray All submit interpreters are in the FREE state.

Overview

Touching the **Submit interpreter** status indicator opens the **All SUBMIT interpreters** overview.

The figure in brackets in the **Selected module** column indicates the current block number in the module.

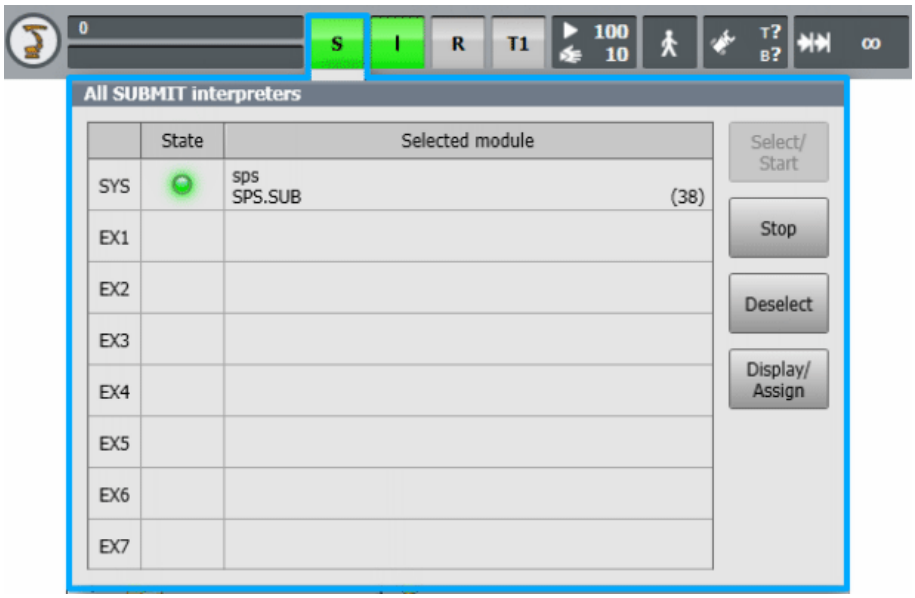


Fig. 12-2: Overview of All SUBMIT interpreters

12.3 Operation of submit interpreter

12.3.1 Automatic starting of submit interpreters (on cold start)

Description

For the extended submits, it is possible to define whether these should start automatically at a cold start, and with which program.

Precondition

- User rights: Function group **General configuration**

Procedure

1. In the status bar, touch the **submit interpreter** status indicator. The **All SUBMIT interpreters** overview opens.

   Select **Display/Assign**.
2. The **Submit interpreter** window opens.

   Select the **Cold start configuration** tab.

   (>>> *12.3.3.2 "“Cold start configuration” tab" Page 580*)
3. In the line of the corresponding extended submit interpreter, activate or deactivate the check box **Autostart** as required.

The settings are immediately valid.

> The default settings for the system submit are:
> - **Available modules** = sps
> - **Autostart** = active
>
> It is strongly recommended not to change these settings.

12.3.2 Manual control of submit interpreters

12.3.2.1 Stopping all submit interpreters

We strongly recommend not deselecting or stopping the system submit interpreter while the robot controller is using functionalities from options (e.g. while a program is running that contains commands from a technology package). If the system submit is not running, this can affect the functionality of technology packages and other options.

Description

This procedure stops all submits that are in the ACTIVE state.

The action has no effect on submits in the STOP, RESET or FREE states.

If a stopped submit is restarted, the SUB program is resumed at the point at which it was stopped.

Precondition

- User rights: Function group **General configuration**
- T1 or T2 mode

Procedure

- In the status bar, touch the **submit interpreter** status indicator. The **All SUBMIT interpreters** overview opens.

  Select **Stop**.

12.3.2.2 Deselecting all submit interpreters

We strongly recommend not deselecting or stopping the system submit interpreter while the robot controller is using functionalities from options (e.g. while a program is running that contains commands from a technology package). If the system submit is not running, this can affect the functionality of technology packages and other options.

Description

This procedure deselects all submits, i.e. both the system submits and all extended submits. All submits are then in the FREE state.

Precondition

- User rights: Function group **General configuration**
- T1 or T2 mode

Procedure

- In the status bar, touch the **submit interpreter** status indicator. The **All SUBMIT interpreters** overview opens.

  Select **Deselect**.

12.3.2.3 Starting all submit interpreters

Description

This procedure starts all submit interpreters that are assigned a SUB program on the **Current display/assignment** tab in the **Submit interpreter** window.

Initial state of SUB program	Effect
STOP	The SUB program is resumed at the point at which it was stopped.
RESET	The SUB program starts from the beginning.
FREE	

Precondition

- User rights: Function group **General configuration**
- T1 or T2 mode
- No submit interpreter is in the ACTIVE state.

Procedure

- In the status bar, touch the **submit interpreter** status indicator. The **All SUBMIT interpreters** overview opens.

  Select **Select/Start**.

12.3.2.4 Stopping, resetting or deselecting individual submit interpreters

We strongly recommend not deselecting or stopping the system submit interpreter while the robot controller is using functionalities from options (e.g. while a program is running that contains commands from a technology package). If the system submit is not running, this can affect the functionality of technology packages and other options.

Precondition

- User rights: Function group **General configuration**
- T1 or T2 mode

Procedure

1. In the status bar, touch the **submit interpreter** status indicator. The **All SUBMIT interpreters** overview opens.

   Select **Display/Assign**.
2. The **Submit interpreter** window opens.

   Select the **Current display/assignment** tab.

   (>>> *12.3.3.1 "Current display/assignment tab" Page 579*)
3. Select the line with the submit interpreter that is to be stopped or deselected.
4. Press the **Stop** or **Deselect** button on the right-hand side.

   If the SUB program is to be reset in the next step, the submit interpreter must be stopped here.
5. If required: now reset the SUB program using the **Reset** button.

   Only possible for submit interpreters that have been stopped, not deselected.

12.3.2.5 Starting individual submit interpreters

Description

The procedure described here can be used both for the system submit and for the extended submits.

For the system submit there are also further ways of starting it individually.

(>>> *12.3.2.6 "Starting the system submit interpreter individually" Page 578*)

Precondition

- User rights: Function group **General configuration**
- T1, T2 or AUT mode
- The submit is in the FREE, STOP or RESET state.

  If a program is to be assigned to the submit: FREE state

Procedure

1. In the status bar, touch the **submit interpreter** status indicator. The **All SUBMIT interpreters** overview opens.

   Select **Display/Assign**.
2. The **Submit interpreter** window opens.

   Select the **Current display/assignment** tab.

   (>>> *12.3.3.1 "Current display/assignment tab" Page 579*)
3. Select the line with the submit interpreter that is to be started.
4. In the **Available modules** column, select the desired SUB program.

   The program SPS.SUB is assigned by default to the system submit interpreter. It is strongly recommended not to change this assignment.

5. Press **Select/Start** or **Start** on the right-hand side.

12.3.2.6 Starting the system submit interpreter individually

Description

The procedures described here start the program sps.sub in the system submit without this having to be selected explicitly. Other submit interpreters cannot be started in this way.

Precondition

- T1, T2 or AUT mode
- The system submit is in the FREE state.

Procedure

- In the Navigator, double-click on the program sps.sub in the directory R1\System.

  Or:
- Select sps.sub in the Navigator and press the **Select** button.

  Or:
- Select sps.sub in the Navigator and select the menu sequence **Edit** > **Select** > **Without parameters**.

  (The menu item **With parameters** is also available, but only **Without parameters** is relevant for SUB programs.)

> If a different SUB program is accidentally started in this way in the system submit interpreter, it must be deselected again and the original sps.sub program intended for the system submit must be selected again.

12.3.3 “Submit interpreter” window

12.3.3.1 Current display/assignment tab

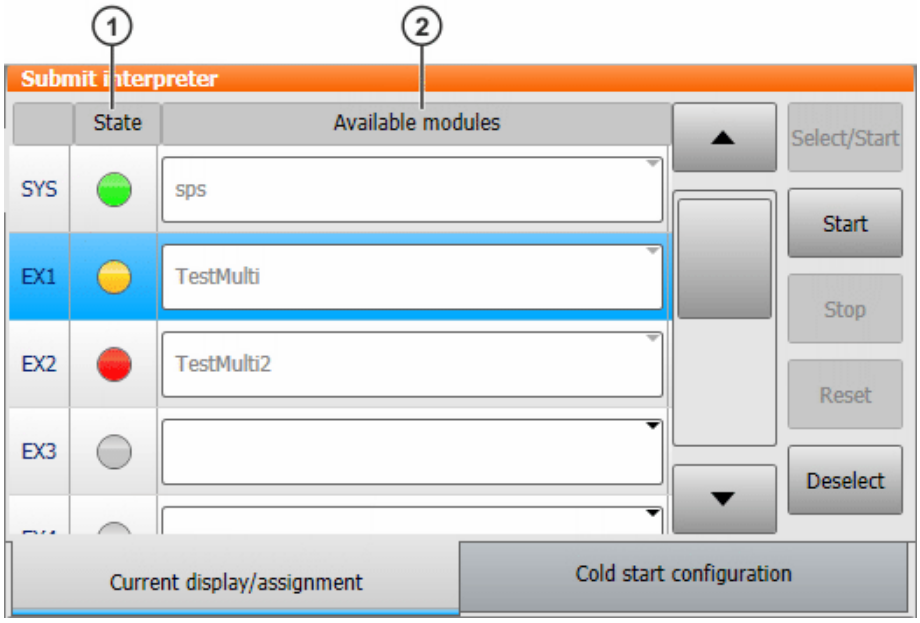


Fig. 12-3: Current display/assignment

Item	Description			
1	State of the interpreter			
		Green ACTIVE		**Yellow** RESET
		Red STOP		**Gray** FREE
2	**SYS** line: The program SPS.SUB is assigned to the system submit interpreter. **EX...** lines: Here a SUB program can be assigned to an extended submit interpreter. The selection list can only be opened if the interpreter is in the FREE state. The list displays SUB programs with the following properties: • Present in the file system under [*Computer name*]\KRC:\ • And: With the module property **Visible** (>>> *"Module info" Page 294*) Programs which are already assigned to an interpreter on this tab, are grayed out in the list and cannot be assigned to any other interpreter.			

By default, the program SPS.SUB is assigned to the system submit interpreter. It is strongly recommended not to change this assignment. It is necessary to enable the controller-internal submit tasks to be executed. If this assignment is changed, this can affect the functionality of technology packages and other options.

If a change is made under **Available modules** and no action is executed for this interpreter using one of the buttons, then a dialog is displayed when the **Submit interpreter** window is closed, asking whether the changes should be saved.

Buttons

Button	Description
Select/Start	Selects the SUB program and starts it. Only available if the program is deselected. **Note:** If there is still an unsaved interpreter <-> module assignment, **Select/Start** saves the assignment, selects the SUB program and starts it. No request for confirmation is generated, asking if the assignment is to be saved.
Start	Starts the SUB program. Only available if the program has been stopped or reset.
Stop	Stops the SUB program. Only available if the program is running.
Reset	Resets the SUB program. Only available if the program has been stopped.
Deselect	Deselects the SUB program. Only available if the program is selected or has been stopped.

12.3.3.2 “Cold start configuration” tab

The settings on this tab are immediately valid and do not have to be saved separately.

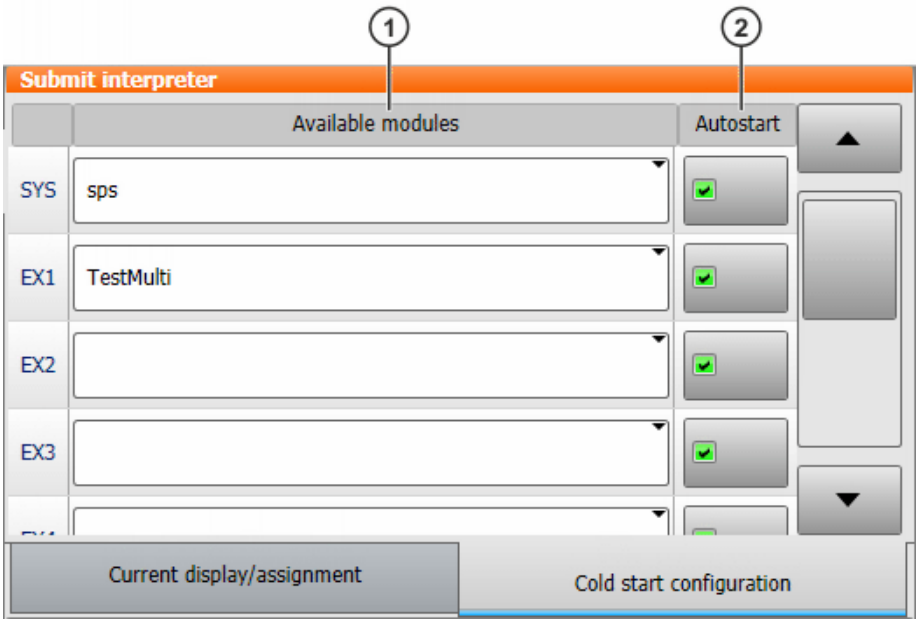


Fig. 12-4: Cold start configuration

Item	Description
1	Here a SUB program can be assigned to an extended submit interpreter. The box displays SUB programs with the following properties: • Present in the file system under [*Computer name*]\KRC:\ • And: With the module property **Visible** (>>> *"Module info" Page 294*) Programs which are already assigned to an interpreter on this tab, are grayed out in the list and cannot be assigned to any other interpreter.
2	• **Activated**: After a cold start, this interpreter starts automatically with the module that is selected under **Available modules**. • **Deactivated**: After a cold start, this interpreter does not start, irrespective of whether a module is selected under **Available modules**.

The default settings for the system submit are:
- **Available modules** = sps
- **Autostart** = active

It is strongly recommended not to change these settings.

12.4 Submit interpreter programming

12.4.1 Creating a new SUB program

Precondition

- User rights: Function group **File operations**
  But at least the user group "**Expert**"

Procedure

1. In the file list, select the folder in which the program is to be created. (Not all folders allow the creation of programs within them.)
2. Press the **New** button.

   The **Template selection** window is opened.
3. Select the template **Submit** or **Expert Submit** and confirm with **OK**.
4. Enter a name for the program and confirm it with **OK**.

Description

“Submit” template:

The **Submit** template generates a SUB file with the following structure:

```
1 DECLARATIONS
2 INI
3
4 LOOP
5 USER PLC
6 ENDLOOP
7 USER SUBROUTINE
```

Line	Description
1	Declaration section
2	Initialization section. For statements that are only to be executed once after the system has booted.
4, 5, 6	LOOP statement containing the Fold USER PLC. USER PLC is for programs that are to run continuously in the background.
7	For user-specific subroutines

“Expert Submit” template:

The **Expert Submit** template generates an empty SUB file. With this template, everything has to be programmed by the user.

Use a LOOP statement when programming. SUB programs without a LOOP statement are only executed once by the Submit interpreter. It is then automatically deselected.

12.4.2 Editing the sps.sub program

Description

The sps.sub program is intended by KUKA for controller-internal submit tasks. User-specific submit tasks should normally be programmed in separate SUB programs.

In some cases, however, it may be necessary for the user to insert instructions into the sps.sub program. The following folds are available for this:

- USER INIT
- USER PLC

Other parts of the sps.sub program must not be modified by the user.

If other parts of sps.sub are changed, this can affect the functionality of technology packages and other options.

Precondition

- The program sps.sub is not selected or has been stopped.
- User group “Expert” or higher

Procedure

1. In the Navigator, select the sps.sub program in the directory R1\System and press **Open**.
2. Enter the changes:
   - Enter initializations in the USER INIT fold. This fold is located in the INI fold.

     ```
     USER INIT
     ; Please insert user defined initialization commands
     ```

   - Enter all other changes in the USER PLC fold.

     ```
     USER PLC
     ; Make your modifications here
     ```

3. Close the program. Respond to the request for confirmation asking whether the changes should be saved by pressing **Yes**.

sps.sub

Structure of the program sps.sub:

```
1 DEF SPS ( )
2 DECLARATIONS
3 INI
4
5 LOOP
6 WAIT FOR NOT($POWER_FAIL)
7 TORQUE_MONITORING()
8
9 ATB PLC LOOP
10 USER PLC
11 ENDLOOP
```

Line	Description
3	INI fold This fold contains the USER INIT fold: here the user can enter statements which are to be executed only once after booting.
5 … 10	LOOP statement. For programs that are to run continuously in the background.
9	Some software options insert folds into the program sps.sub. Example: KUKA.ArcTech Basic inserts the fold `ATB PLC LOOP`. The folds that are actually present depend on what options are installed on the robot controller.
10	USER PLC: Here the user can enter instructions that are to be executed in the LOOP.

12.4.3 KRL instructions not allowed or subject to restrictions

Limitations

Almost all KRL instructions can be used in a SUB program. The following instructions are not possible, however – or only with restrictions:

- Instructions for robot motions

  Robot motions can only be interpreted by the robot interpreter. For this reason, SRC programs containing motion commands cannot be called as subprograms from a SUB program.
- Instructions referring to robot motions

  These include BRAKE and all TRIGGER statements.
- The following instructions are permissible in the system submit, but not in the extended submits:
  - ASYPTP
  - VECTORMOVEON()
  - VECTORMOVEOFF()
  - SET_TORQUE_LIMITS()
  - RESET_TORQUE_LIMITS()

Example

The motion instructions for external axes in this example can be used in a SUB program provided that this is assigned to the system submit and not to an extended submit.

```
IF (($IN[12] == TRUE) AND ( NOT $IN[13] == TRUE)) THEN
$VEL_EXTAX[2]=10
$ACC_EXTAX[2]=10
ASYPTP {E2 45}
...
IF ((NOT $IN[12] == TRUE) AND ($IN[13] == TRUE)) THEN
$VEL_EXTAX[2]=10
$ACC_EXTAX[2]=10
ASYPTP {E2 0}
```

External axis E2 is moved in accordance with specific inputs.

12.4.4 Subprograms

Other programs can be called as subprograms in a SUB program. The following are possible:

- Other SUB programs
- SRC programs without statements for robot motions

Example:

CELL.SRC can be selected from within the program sps.sub with a CWRITE statement and RUN. The call only takes effect in the case of a cold start.

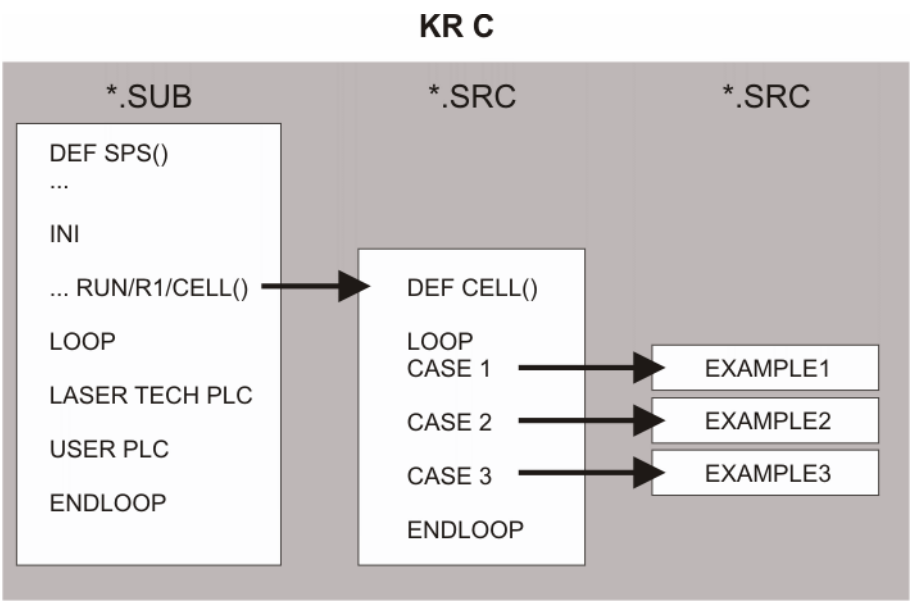


Fig. 12-5: sps.sub selects CELL.SRC in the robot interpreter

12.4.5 Inputs/outputs and communication

Inputs/outputs

Submit interpreters can access the inputs and outputs of the robot controller.

WARNING

No monitoring is carried out as to whether the robot interpreter and submit interpreter are accessing the same output simultaneously, as this may even be desired in certain cases.
The user must therefore carefully check the assignment of the outputs. Otherwise, unexpected output signals may be generated, e.g. in safety equipment. Death, serious injuries or major damage to property may result.

Communication

The flags of the robot controller can be used to enable the exchange of binary information between a running motion program and a SUB program. A flag is set by the submit interpreter and read by the robot interpreter.

12.4.6 System variables

The submit interpreters have read-access to all system variables and write-access to many of them. Access works even if the system variables are being used in parallel by a motion program.

If a system variable to which the submit interpreter does not have write-access is modified in a SUB program, an error message is generated when the SUB program is started and the submit interpreter stops.

WARNING

In the test modes, $OV_PRO must not be written to by a submit interpreter, because the change may be unexpected for operators working on the industrial robot. Death, injuries or damage to property may result.

WARNING

If possible, do not modify safety-relevant signals and variables (e.g. operating mode, EMERGENCY STOP, safety gate contact) via a submit interpreter.
If modifications are nonetheless required, all safety-relevant signals and variables must be linked in such a way that they cannot be set to a dangerous state by the submit interpreter or PLC.

12.4.6.1 Changes in relation to Single Submit mode

In KSS, a system submit interpreter and 7 extended submit interpreters are available. This is called “Multi-Submit mode”. In earlier versions of the KSS, only the system submit interpreter was available as standard. This is called “Single Submit mode”.

In Multi-Submit mode, the meaning of certain system variables is changed compared with the Single Submit mode. Beyond this, system variables have been added and others have been removed.

Changed system variables

Changed system variables:

- $INTERPRETER

  The variable has more possible values than beforehand. The integer value representing the system submit interpreter has changed.

  (>>> *12.4.6.2 "$INTERPRETER" Page 587*)
- $PRO_STATE0

  The variable previously indicated the program status of the submit interpreter. It now indicates the status of the group indicator in the status bar. The variable is write-protected.

  (>>> *12.2 "Status indicator for the submit interpreter" Page 574*)

  The states that have been provided by the variables $PRO_*xxx*0 in Single Submit mode are provided in Multi-Submit mode by $PROG_INFO[].
- $ERR

  The component `interpreter` has more possible values than beforehand.

  (>>> *11.7.8.4 "$ERR" Page 498*)

Removed system variables

Removed system variables (with the postfix “0”):

- $PRO_IP0
- $PRO_NAME0[]
- $PRO_MODE0
- $WAIT_FOR_ON0
- $WAIT_FOR0[]

The states that have been provided by the variables $PRO_*xxx*0 in Single Submit mode are provided in Multi-Submit mode by $PROG_INFO[].

The system variables of the same name with the postfix “1” ($PRO_IP1, etc.) are retained unchanged.

The variables $WAIT_*xxx*0 have been removed altogether.

New system variable

New system variable:

- $PROG_INFO[] (>>> 12.4.6.3 "$PROG_INFO[]" Page 587)

12.4.6.2 $INTERPRETER

Description

Selecting the interpreter

Numerous system states can be read, and in many cases also set, via variables. Strictly speaking, these variables exist multiple times – once per interpreter. They are identical in name for all interpreters.

Examples:

- Program run mode ($PRO_MODE)
- Program state ($PRO_STATE)
- Data of the process pointer ($PRO_IP)
- WAIT FOR statement at which the interpreter is currently waiting ($WAIT_FOR)

When such a variable is accessed, this is always referred automatically to the current interpreter. This is defined by $INTERPRETER.

Syntax

`$INTERPRETER=`*Interpreter*

Explanation of the syntax

Element	Description
Interpreter	Type: INT Selecting the interpreter • 1: Robot interpreter • 2: System submit interpreter • 3: Extended submit interpreter 1 • 4: Extended submit interpreter 2 • Etc. Default: 1

12.4.6.3 $PROG_INFO[]

Description

$PROG_INFO[] groups certain system states together in a structure. The states refer to an interpreter.

Syntax

`$PROG_INFO[`*Interpreter*`] = `*Information*

Explanation of the syntax

Element	Description
Interpreter	Type: INT • 1: Robot interpreter • 2: System submit interpreter • 3: Extended submit interpreter 1 • 4: Extended submit interpreter 2 • Etc.
Information	Type: Prog_Info List of the system states, referred to *Interpreter*

Prog_Info

```
STRUC Prog_Info CHAR sel_name[32], PRO_STATE p_state,
PRO_MODE p_mode, CHAR pro_ip_name[32], INT pro_ip_snr
```

Element	Description
sel_name[]	Name of the selected program
p_state	Program status • #P_ACTIVE: Program is selected and is running. • #P_FREE: Program is deselected. • #P_RESET: Program is selected and has been stopped and reset. • #P_STOP: Program is selected and has been stopped.
p_mode	Program run mode • #GO • #MSTEP • #ISTEP • #BSTEP • #PSTEP • #CSTEP
pro_ip_name[]	Name of the current module
pro_ip_snr	Current block in the current module

Example

```
DEF myProgr()
...
WAIT FOR $PROG_INFO[4].P_STATE == #P_ACTIVE
```

Meaning: Wait until extended submit 2 has selected and started a program.

12.4.7 CWRITE: changes in relation to Single Submit mode

CWRITE

CWRITE can transfer statements to an interpreter via the command channel $CMD. The meaning of some commands in Multi-Submit mode has changed compared with Single Submit mode.

Changed commands

The meaning of the following commands has changed compared with the Single Submit mode:

- RUN [*Interpreter ID*]
- STOP [*Interpreter ID*]
- RESET [*Interpreter ID*]
- CANCEL [*Interpreter ID*]

Interpreter ID:

- 0: All submit interpreters
- 1: Robot interpreter
- 2: System submit interpreter
- 3: Extended submit interpreter 1
- 4: Extended submit interpreter 2
- Etc.

In addition, RUN has been expanded with the optional element [> *Interpreter ID*].

Example 1

```
CWRITE ($CMD, STAT, MODE, "RUN/R1/CELL()")
```

Behavior in Single Submit mode:
Starts the program CELL(). Since CELL() is an SRC program, it starts in the robot interpreter.

Behavior in Multi-Submit mode:
As in Single Submit mode. This program line can be used in the system submit or in an extended submit.

Example 2

```
CWRITE ($CMD, STAT, MODE, "RUN/R1/SPS()")
```

Behavior in Single Submit mode:
Starts the program SPS(). Since SPS() is a SUB program, it starts in the system submit interpreter (= only submit interpreter in Single Submit mode).

Behavior in Multi-Submit mode:
Starts the program SPS(). Since SPS() is a SUB program, it starts in the system submit interpreter.

Example 3

```
CWRITE ($CMD, STAT, MODE, "STOP 0")
```

This program line is only meanful in a robot program.

Behavior in Single Submit mode:
Stops the system submit interpreter.

Behavior in Multi-Submit mode:
Stops all running submit interpreters.

Example 4

```
CWRITE ($CMD, STAT, MODE, "CANCEL 0")
```

Behavior in Single Submit mode:
Deselects the system submit interpreter.

Behavior in Multi-Submit mode:
Deselects all submit interpreters.

Example 5

```
CWRITE ($CMD, STAT, MODE, "RUN/R1/MySubProg() > 5")
```

This statement is not permissible in Single Submit mode.

Behavior in Multi-Submit mode:
If MySubProg() is a SUB program, it starts in extended submit 3. This program line can be used in the other submit interpreters or in a robot program.

Example 6

```
CWRITE ($CMD, STAT, MODE, "STOP 5")
```

This statement is not permissible in Single Submit mode.

Behavior in Multi-Submit mode:
Stops extended submit interpreter 3. This program line can be used in the other submit interpreters or in a robot program.

Example 7

```
CWRITE ($CMD, STAT, MODE, "CANCEL 5")
```

This statement is not permissible in Single Submit mode.

Behavior in Multi-Submit mode:
Deselects extended submit interpreter 3. This program line can be used in the other submit interpreters or in a robot program.

12.5 Overview: changes of state due to actions in the menu

The tables show how submit interpreters with different initial states change their state as a result of specific actions.

Example 1:

Initial state	1st action Stop	2nd action Select/Start	3rd action Deselect	4th action Select/Start
ACTIVE	STOP	ACTIVE	FREE_PRESEL.	ACTIVE
STOP	STOP	ACTIVE	FREE_PRESEL.	ACTIVE

Initial state	1st action Stop	2nd action Select/Start	3rd action Deselect	4th action Select/Start
RESET	RESET	ACTIVE	FREE_PRESEL.	ACTIVE
FREE_PRESEL.	FREE_PRESEL.	ACTIVE	FREE_PRESEL.	ACTIVE
FREE_UNSEL.	FREE_UNSEL.	FREE_UNSEL.	FREE_UNSEL.	FREE_UNSEL.

Example 2:

Initial state	1st action Stop	2nd action Deselect
ACTIVE	STOP	FREE_PRESEL.
STOP	STOP	FREE_PRESEL.
RESET	RESET	FREE_PRESEL.
FREE_PRESEL.	FREE_PRESEL.	FREE_PRESEL.
FREE_UNSEL.	FREE_UNSEL.	FREE_UNSEL.

Example 3:

Initial state	1st action Deselect
ACTIVE	FREE_PRESEL.
STOP	FREE_PRESEL.
RESET	FREE_PRESEL.

Initial state	1st action Deselect
FREE_PRESEL.	FREE_PRESEL.
FREE_UNSEL.	FREE_UNSEL.

13 Diagnosis

13.1 Logbook

13.1.1 Displaying the logbook

The operator actions on the smartPAD are automatically logged.

Precondition

- User rights: Function group **Diagnostic functions**

Procedure

- In the main menu, select **Diagnosis** > **Logbook** > **Display**.

The following tabs are available:

- Log (>>> *13.1.2 ""Log" tab" Page 593*)
- Filter (>>> *13.1.3 "Filter tab" Page 595*)

13.1.2 “Log” tab

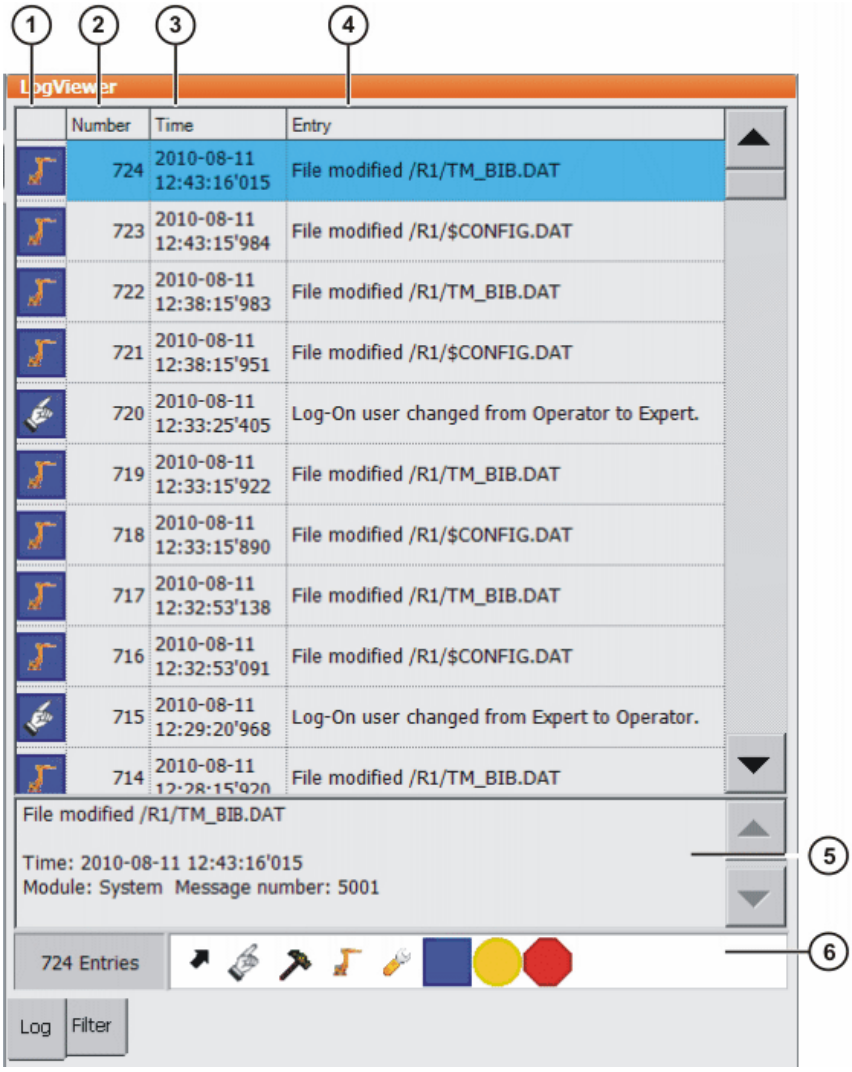


Fig. 13-1: Logbook, Log tab

Item	Description
1	Type of log event Example : Filter type "Information" + filter class "System" = information originated by the kernel system of the robot. The individual filter types and filter classes are listed on the **Filter** tab.
2	Log event number
3	Date and time of the log event
4	Brief description of the log event
5	Detailed description of the selected log event
6	Indication of the active filter

The following buttons are available:

Button	Description
Export	Exports the log data as a text file. (>>> *13.1.4 "Configuring the logbook" Page 595*) Required user rights: Function group **Archive to local HDD/SSD**
Refresh	Refreshes the log display.

13.1.3 Filter tab

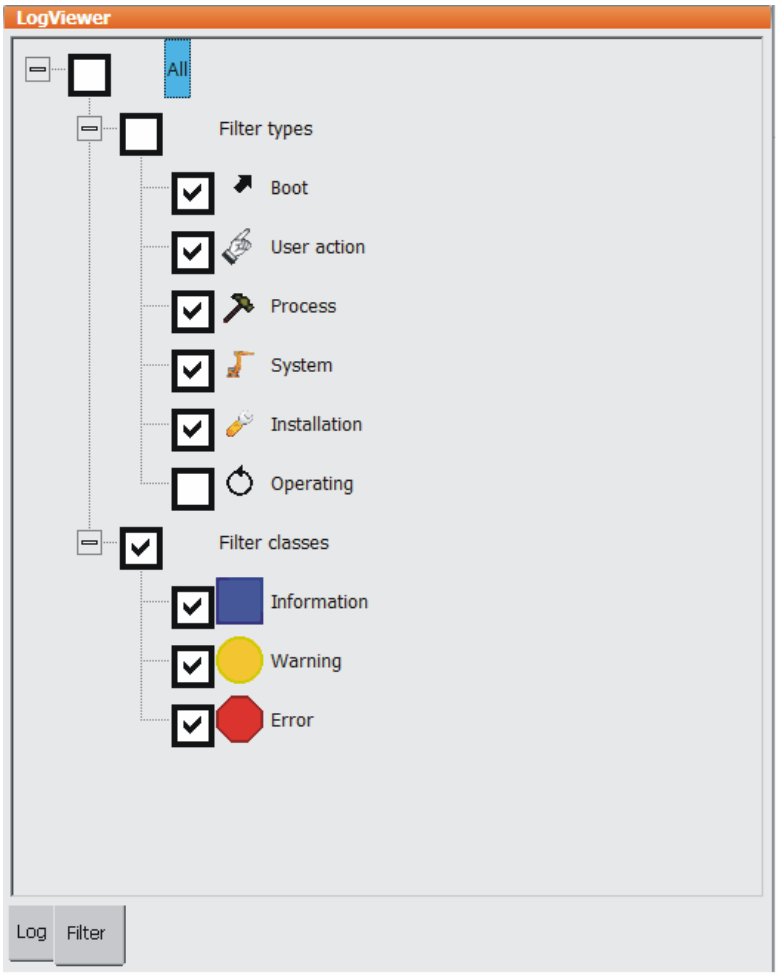


Fig. 13-2: Logbook, Filter tab

13.1.4 Configuring the logbook

Precondition

- User rights: Function group **General configuration**

Procedure

1. In the main menu, select **Diagnosis** > **Logbook** > **Configuration**. A window opens.
2. Make the desired settings.
3. Press **OK** to save the configuration and close the window.

Description

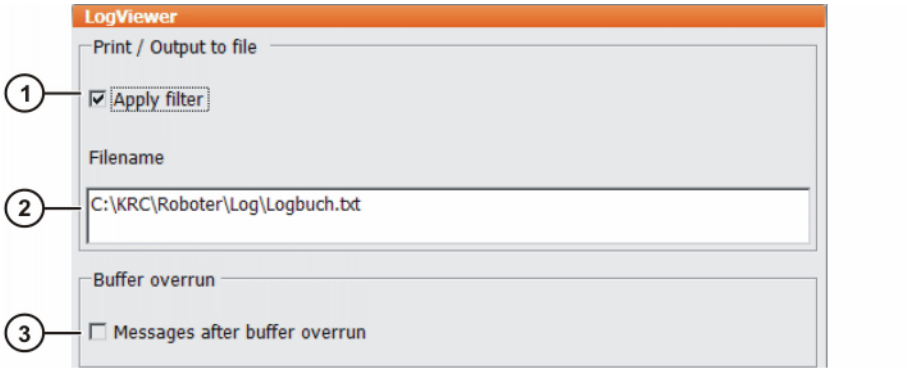


Fig. 13-3: Logbook configuration window

Item	Description
1	• **Check box active**: the log events selected with the filter are saved in the text file. • **Check box not active**: all log events are saved in the text file.
2	Enter the path and name of the text file. Default path: C:\KRC\ROBOTER\LOG\LOGBUCH.TXT
3	• **Check box active**: log data deleted because of a buffer overflow are indicated in gray in the text file. • **Check box not active**: log data deleted because of a buffer overflow are not indicated in the text file.

13.2 Displaying the caller stack

This function displays the data for the process pointer ($PRO_IP).

Precondition

- User rights: Function group **Diagnostic functions**
- A program is selected.

Procedure

- In the main menu, select **Diagnosis** > **Caller stack**.

Description

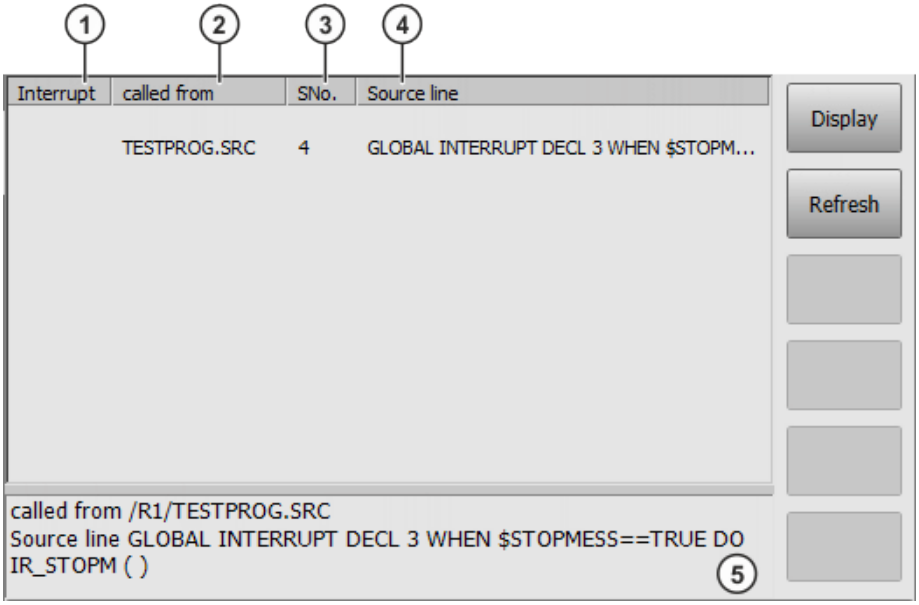


Fig. 13-4: “Caller stack” window

Item	Description
1	• [With info]: call initiated by an interrupt. • [Empty]: call not initiated by an interrupt.
2	This file contains the call.
3	The program line with this number contains the call. Preconditions in the program for the correct line to be determined using the number: • Detail view is activated. • All Folds are open.
4	Source line
5	Detailed information about the entry selected in the list

13.3 Displaying interrupts

Precondition

- User rights: Function group **Diagnostic functions**

Procedure

1. In the main menu, select **Diagnosis** > **Interrupts**. The **Interrupts** window is opened.
2. In the box at bottom left, select the interpreter whose interrupts are to be displayed.
3. The states of the interrupts are displayed. The display is not refreshed automatically, however.

   To refresh the display, press **Refresh**.

“Interrupts” window

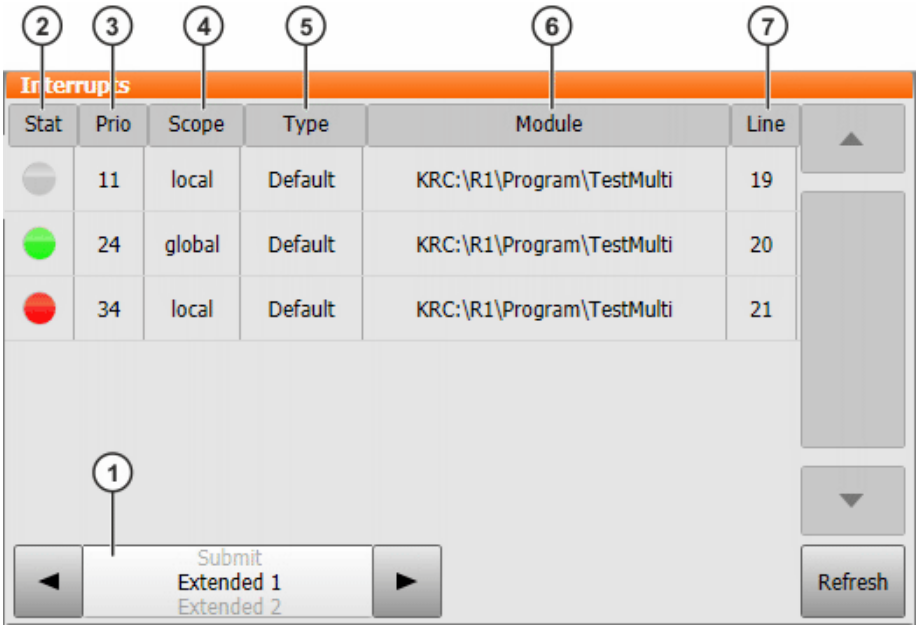


Fig. 13-5: Interrupts

Item	Description
1	Here you can select the interpreter whose interrupts are to be displayed.
2	State: • GREEN: “Interrupt ON” or “Interrupt ENABLE” • RED: “Interrupt DISABLE” • GRAY: “Interrupt OFF” or: Interrupt only declared, but not ON
3	Number/priority of the interrupt
4	Validity range of the interrupt: global or local
5	Type of interrupt, dependent on the defined event in the interrupt declaration • **Standard**: e.g. $IN[...] • **Error stop**: $STOPMESS • **EMERGENCY STOP**: $ALARM_STOP • **Measurement (Fast Measurement)**: $MEAS_PULSE[1…5] • **Trigger**: Trigger subprogram
6	Module in which the interrupt is declared **Note:** If the path is very long and cannot be shown completely in the box, it can be displayed in its entirety by touching the box.
7	Program line in which the interrupt is declared

Button	Description
Refresh	Refreshes the display.

13.4 Displaying diagnostic data about the kernel system

Description

The menu item **Diagnostic monitor** makes it possible to display a wide range of diagnostic data concerning numerous software sub-areas of the kernel system.

Examples:

- Area **Kcp3 driver** (= driver for the smartPAD)
- Network driver

The data displayed depend on the selected area. The display includes states, fault counters, message counters, etc.

Precondition

- User rights: Function group **Diagnostic functions**

Procedure

1. In the main menu, select **Diagnosis** > **Diagnostic monitor**.
2. Select an area in the **Module** box.
   Diagnostic data are displayed for the selected area.

13.5 Automatically compressing data for error analysis (KRCDiag)

Description

If it is necessary for an error to be analyzed by KUKA Support, this procedure can be used to compress the required data. The procedure generates a ZIP file in the directory C:\KUKA\KRCDiag. This contains the data which KUKA Support requires to analyze an error. This includes information about the system resources, screenshots and much more.

Preparation

A screenshot of the current view of the smartHMI is automatically generated for the data packet.

- Therefore, if possible, display the information related to errors on the smartHMI before starting the procedure:
  e.g. expand the message window or display the logbook. What information is useful here depends on the specific circumstances.

Procedure via “Diagnosis”

Required user rights: Function group **Diagnostic functions**

- In the main menu, select **Diagnosis** > **KrcDiag**.
  The data are compressed. Progress is displayed in a window. Once the operation has been completed, this is also indicated in the window. The window is then automatically hidden again.

Procedure via smartPAD

This procedure uses keys on the smartPAD instead of menu items. It can thus also be used if the smartHMI is not available, due to Windows problems for example.

Precondition:

- The smartPAD is connected to the robot controller.
- The robot controller is switched on.

The keys must be pressed within 2 seconds. Whether or not the main menu and keypad are displayed in the smartHMI is irrelevant.

1. Press the “Main menu” key and hold it down.
2. Press the keypad key twice.
3. Release the “Main menu” key.

   The data are compressed. Progress is displayed in a window. Once the operation has been completed, this is also indicated in the window. The window is then automatically hidden again.

Procedure via “Archive”

Alternatively, the data can also be compressed via **File** > **Archive** > **[...]** > **KrcDiag**. In this way, the data can be stored on a USB stick or network path.

(>>> *7.10 "Archiving and restoring data" Page 307*)

14 Installation

The robot controller is supplied with a Windows operating system and an operational version of the KUKA System Software (KSS). Therefore, no installation is required during initial start-up.

Installation becomes necessary, for example, in the event of the hard drive being damaged and exchanged.

The robot controller may only be operated using the software provided with the controller by KUKA.
KUKA Deutschland GmbH must be consulted if different software is to be used. (>>> *16 "KUKA Service" Page 623*)

14.1 System requirements

The System Software 8.5 can be run on the following robot controller:

- KR C4
- with Windows Embedded Standard 7 V4.x
- and with at least 2 GB RAM

14.2 Installing additional software

This function can be used to install additional software, e.g. technology packages. New programs and updates can be installed. The software is installed from a USB stick. Alternatively, it can also be installed via a network path.

Precondition

- User rights: Function group **General configuration**
- T1 or T2 mode
- No program is selected.
- USB stick with the software to be installed
  - ZIP files must be unzipped.
  - There must be no other files in the directory in which the individual files are located.

NOTICE

We recommend using a KUKA USB stick. Data may be lost if a stick from a different manufacturer is used.

Depending on the individual additional software, the procedure may deviate from the following general description. The installation description for the specific software can be found in the corresponding documentation.

Procedure

1. Connect the USB stick to the robot controller or smartPAD.
2. In the main menu, select **Start-up** > **Additional software**.
3. Press **New software**: The new software must be displayed in the **Name** column and drive **E:** or **K:** in the **Path** column.

   If not, press **Refresh**.
4. If the specified entries are now displayed, continue with step 5.

Otherwise, the path from which the software is to be installed must be configured first:

a. Press the **Configure** button.
b. Select a line in the **Installation paths for options** area.

   Note: If the line already contains a path, this path will be overwritten.
c. Press **Path selection**. The available drives are displayed.
d. If the stick is connected to the robot controller: On **E:** navigate to the directory with the software. Select the directory.

   If the stick is connected to the smartPAD: **K:** instead of **E:**
e. Press **Save**. The **Installation paths for options** area is displayed again. It now contains the new path.
f. Mark the line with the new path and press **Save** again.

5. Select the new software and press **Install**. Answer the request for confirmation with **Yes**.
6. Confirm the reboot prompt with **OK**.
7. Remove the stick.
8. Reboot the robot controller.

Description

The following buttons are available:

Button	Description
New software	All programs available for installation are displayed.
Back	Additional software already installed is displayed.
Refresh	Refreshes the display, e.g. after a USB stick has been connected.
Install	Displays additional buttons: • **Yes**: The selected software is installed. If it is necessary to reboot the controller, this is indicated by a message. • **No**: The software is not installed.
Configure	This button is only displayed if **New software** has been pressed. Paths for the installation of additional software or for updates of the System Software can be selected and saved here. Displays additional buttons: • **Path selection**: A new path can be selected. • **Save**: Saves the displayed paths.
Uninstall	Displays additional buttons: • **Yes**: The selected software is uninstalled. • **No**: The software is not uninstalled.

14.3 KSS update

Description

This function can be used to install KSS updates, e.g. from KSS 8.5.0 to KSS 8.5.1.

Following installation or update of the KUKA System Software, the robot controller always performs an initial cold start.

It is advisable to archive all relevant data before updating a software package. If necessary, the old version can be restored in this way. It is also advisable to archive the new version after carrying out the update.

Do not use this function to install a new version, e.g. from KSS 8.3 to KSS 8.5. KUKA Deutschland GmbH must be consulted before a new version or variant is installed.
This function cannot be used to install updates of additional software, such as technology packages.

Overview

There are 2 ways of installing a KSS update:

- From USB memory stick
  (>>> *14.3.1 "Update from USB stick" Page 603*)
- From the network
  (>>> *14.3.2 "Update from the network" Page 604*)

14.3.1 Update from USB stick

NOTICE

A non-bootable USB stick must be used.
We recommend using a non-bootable KUKA stick. Data may be lost if a stick from a different manufacturer is used.

Precondition

- User rights: Function group **General configuration**
- T1 or T2 mode
- No program is selected.
- USB stick with the software to be installed
  - ZIP files must be unzipped.
  - There must be no other files in the directory in which the individual files are located.

Procedure

1. Plug in the USB stick.
2. In the main menu, select **Start-up** > **Software update** > **Automatic**.
3. A request for confirmation is displayed, asking if the update should be carried out. Confirm by pressing **Yes**.

   The data are copied locally. The progress is displayed in the message window.

   When the process is finished, the following message is displayed:
   Preparation for Setup is finished. Please reboot the controller to execute the Setup.

4. Select **Shutdown** in the main menu and then the option **Reboot control PC**. (**Reload files** is not required.)
5. Confirm the request for confirmation with **Yes**. The robot controller is rebooted and performs the update.

   The robot controller then reboots again.

The updated System Software is now available.

14.3.2 Update from the network

Description

In the case of an update from the network, the installation data are copied to the local drive D:\. If there is already a copy of a system software version present on D:\, that copy will now be overwritten.

Installation is started on completion of the copying operation.

Precondition

- User rights: Function group **General configuration**
- T1 or T2 mode
- No program is selected.

Preparation

Configure the network path from which the update installation is to be carried out:

1. In the main menu, select **Start-up** > **Additional software**.
2. Press **New software**.
3. Press **Configure**.
4. Select the **Installation path for KRC update via the network** box. Press **Path selection**.
5. Select the desired network path (= the directory in which the Setup.exe file is located). Press **Save**.
6. The selected path is now displayed in the **Installation path for KRC update via the network** box.

   Press **Save** again.
7. Close the window.

> It is only necessary to configure the network path once. It remains saved for subsequent updates.

Procedure

1. In the main menu, select **Start-up** > **Software update** > **Net**.
2. A request for confirmation is displayed, asking if the update should be carried out. Confirm by pressing **Yes**.

   The data are copied locally. The progress is displayed in the message window.

   When the process is finished, the following message is displayed: *Preparation for Setup is finished. Please reboot the controller to execute the Setup.*
3. Select **Shutdown** in the main menu and then the option **Reboot control PC**. (**Reload files** is not required.)
4. Confirm the request for confirmation with **Yes**. The robot controller is rebooted and performs the update.

The robot controller then reboots again.

The updated System Software is now available.

14.4 Automatically updating the firmware of hardware components

Description

If a KUKA hardware component has been exchanged or added to the system, the system software detects if this hardware has out-of-date firmware. It prepares an automatic update and displays the following message: *An automatic firmware update will be started in {Countdown} seconds.*

The counter starts at 60 seconds. While it is running, the user has the option of aborting the process and starting it manually at a later time.

Properties of the firmware detection:

- Detection is active if T1 is selected and no program is selected.

  If these preconditions are met, the firmware is detected as soon as the hardware is present in the system. No reboot is required for the detection process.
- The system software has current firmware for KUKA hardware components. This is used to update the hardware. No network connection is required.
- If there is new firmware available for multiple hardware components, everything is updated at once. It is not possible to update components individually.
- No user input may be entered during the smartPAD update. The update can take up to 2 minutes per hardware component.
- No reboot of the system is required following the update.

Precondition

Only if the update is to be started manually:

- Administrator user group

Procedure

Do not switch off the robot controller during the update.

Start the update immediately:

1. Press **Start** in the window with the message.

   Or: Wait until the second counter has elapsed. The update then starts automatically.
2. Observe the message window: It displays messages about the status of the update. Furthermore, the dialog window shows a progress bar.

   Once the update has been completed successfully, the following message is displayed for each component:

   {Name of the component} has been successfully updated to version {Version of the firmware in the format 1.1.0-1}.

Start the update later (manually):

1. Press **Cancel** in the window with the message.
2. At the desired subsequent point in time, select **Start-up** > **Service** > **Firmware / Hardware Manager** in the main menu.

   The **Firmware / Hardware Manager** window opens. The affected hardware components are indicated by **Update available**.

3. Press **Update all**. The update starts.
4. Observe the message window: It displays messages about the status of the update. Furthermore, the dialog window shows a progress bar.

   Once the update has been completed successfully, the following message is displayed for each component:

   {Name of the component} has been successfully updated to version {Version of the firmware in the format 1.1.0-1}.

Fault/power failure

An update cannot damage a hardware component, even if it fails, e.g. due to a power failure. A failed update can be corrected by performing it again.

If a power failure occurred during an update and it was thus not possible to load the firmware completely onto the hardware, the robot controller indicates this by means of error messages. The user can start the update again in the **Firmware / Hardware Manager** window. The robot controller then performs a cold restart.

“Firmware / Hardware Manager” window

The **Firmware / Hardware Manager** window contains a list of the EtherCAT devices.

- For each device, it specifies whether the firmware is current or whether an update is available.
- When expanded, the list entries display information about the devices:
  - Article number
  - Serial number
  - Hardware version
  - Firmware version

If **Update available** is displayed for one or more devices, the update can be started manually via **Update all**. (Only possible for the “Administrator” user group.)

15 Appendix

15.1 Assignment of functions and function groups

Information about the tables

The tables in the following sections provide detailed information about which system software functions belong to which function groups. The assignment of the functions to the function groups cannot be changed.

- The entry in the **Function group** column refers to the operator control element specified in the left-hand column, without its subfunctions.
- Operator control elements can have subfunctions (e.g. a window opens, a command selection opens, etc.).

  The subfunctions may belong to different function groups than the higher-level element. The subfunctions are thus specified separately.
- Entry “None” in the **Function group** column:

  These functions are not assigned to any function group. Any user may execute them. This cannot be changed.

Unchangeable rights

Some functions of the System Software are not assigned to any function group and only defined user groups may execute them. These definitions cannot be modified. These functions include:

Function	User group
Assigning a user group to a function group (under **Start-up** > **Rights management**)	**Administrator**
Activating projects that contain changes to the safety configuration	**Safety maintenance technician** or higher
Activating projects that contain no changes to the safety configuration	**Expert**
Importing a safety configuration	**Safety maintenance technician** or higher
Editing safety functions (various)	**Safety recovery technician** or **Safety maintenance technician** or higher **Note:** The required rights are specified in the descriptions of the individual functions.
Creating a program as **User**: no template can be selected. Creating a program as **Expert** or higher: a template can be selected.	In the function group **File operations**, it is possible to configure which user group can create programs. The right to select templates is invariably linked, however, to the user group **Expert** or higher.

15.1.1 Menu item File

Menu sequence	Function group
File > **Print** > **Logbook**	**Print functions**
File > **Archive** > **USB (KCP)** > **All**	**Archive to USB drives**

Menu sequence	Function group
File > Archive > USB (KCP) > Applications	**Archive to USB drives & Partial archiving**
File > Archive > USB (KCP) > System data	
File > Archive > USB (KCP) > Log data	
File > Archive > USB (KCP) > KrcDiag	None
File > Archive > USB (cabinet) > All	**Archive to USB drives**
File > Archive > USB (cabinet) > Applications	**Archive to USB drives & Partial archiving**
File > Archive > USB (cabinet) > System data	
File > Archive > USB (cabinet) > Log data	
File > Archive > USB (cabinet) > KrcDiag	None
File > Archive > Network > All	**Archive to network**
File > Archive > Network > Applications	**Archive to network & Partial archiving**
File > Archive > Network > System data	
File > Archive > Network > Log data	
File > Archive > Network > KrcDiag	**Archive to network**
File > Archive > Logbook	**Archive to local HDD/SSD**
File > Restore > USB (KCP) > All	**Restore**
File > Restore > USB (KCP) > Applications	**Partial restoration**
File > Restore > USB (KCP) > System data	
File > Restore > USB (cabinet) > All	**Restore**
File > Restore > USB (cabinet) > Applications	**Partial restoration**
File > Restore > USB (cabinet) > System data	
File > Restore > Network > All	**Restore**
File > Restore > Network > Applications	**Partial restoration**
File > Restore > Network > System data	
File > Backup Manager > Backup configuration	**General configuration**
File > Backup Manager > Back up	**Archive with unknown destination**
File > Backup Manager > Save as...	
File > Backup Manager > Restore > Projects and options	**Restore**
File > Backup Manager > Restore > RDC data	**Restoration of critical data**
File > Backup Manager > Restore from ... > Projects and options	**Restore**
File > Backup Manager > Restore from ... > RDC data	**Restoration of critical data**

15.1.2 Menu item Configuration

Menu sequence	Function group
Configuration > Inputs/outputs > Automatic External Subfunctions: (>>> *15.1.2.1 "Automatic External" Page 609*)	None
Configuration > Inputs/outputs > I/O drivers Subfunctions: (>>> *15.1.2.2 "I/O drivers" Page 610*)	None
Configuration > SUBMIT interpreter	**General configuration**

Menu sequence	Function group
Configuration > **Status keys**	None
Configuration > **User group**	None
Configuration > **Miscellaneous** > **Language**	**General configuration**
Configuration > **Miscellaneous** > **Workspace monitoring** > **Override**	**General configuration**
Configuration > **Miscellaneous** > **Workspace monitoring** > **Configuration** Subfunctions: (>>> *15.1.2.3 "Cartesian workspaces/Axis-specific workspaces" Page 610*)	None
Configuration > **Miscellaneous** > **Event planner** Subfunctions: (>>> *15.1.2.4 "Event planner" Page 610*)	**General configuration**
Configuration > **Miscellaneous** > **Point coordinate correction limit** Subfunctions: (>>> *15.1.2.5 "Point coordinate correction limit" Page 610*)	None
Configuration > **Safety configuration** **Note:** The user rights required for the subfunctions are independent of the function groups. They are permanently defined and cannot be changed. What right is required for what subfunction is specified in the description of the corresponding function.	None
Configuration > **Brake test configuration** Subfunctions: User group "**Safety maintenance technician**" or higher	None
Configuration > **Machine configuration** Subfunctions: **General configuration**	**General configuration**
Configuration > **Collision detection** > **View**	None
Configuration > **Collision detection** > **Jogging configuration** Subfunctions: **Calibration**	None
Configuration > **Collision detection** > **Data set configuration** Subfunctions: **Calibration**	None

15.1.2.1 Automatic External

Operator control element	Function group
Toggles between the pages via: **Display/Configure/Inputs/Outputs**	None
Normal/Details	None
Edit	**General configuration**
OK	**General configuration**
Cancel	None

15.1.2.2 I/O drivers

Operator control element	Function group
Reconfigure	**General configuration**

15.1.2.3 Cartesian workspaces/Axis-specific workspaces

Operator control element	Function group
Toggles between the pages via: **Signal/Cartesian/Axis-specific**	None
Save	**General configuration**

15.1.2.4 Event planner

Operator control element	Function group
Refresh	None
Save	**General configuration**

15.1.2.5 Point coordinate correction limit

Operator control element	Function group
Correction limit active	**General configuration**
Cartesian distance	
Angle of rotation	

15.1.3 Menu item Display

Menu sequence	Function group
Display > **Inputs/outputs** > **Digital inputs**	**General configuration** Subfunctions: **General configuration**
Display > **Inputs/outputs** > **Digital outputs**	
Display > **Inputs/outputs** > **Analog inputs**	
Display > **Inputs/outputs** > **Analog outputs**	
Display > **Inputs/outputs** > **Automatic External** Subfunctions: (>>> *15.1.2.1 "Automatic External" Page 609*)	None
Display > **Inputs/outputs** > **I/O drivers** Subfunctions: (>>> *15.1.2.2 "I/O drivers" Page 610*)	None
Display > **Actual position**	None
Display > **Variable** > **Single** Subfunctions: (>>> *15.1.3.1 "Variable display" Page 611*)	None
Display > **Variable** > **Overview** > **Display** Subfunctions: (>>> *15.1.3.2 "Variable overview" Page 611*)	None

Menu sequence	Function group
Display > Variable > Overview > Configuration Subfunctions: (>>> *15.1.3.2 "Variable overview" Page 611*)	**General configuration**
Display > Variable > Overview > Edit "ConfigMon.ini"	**General configuration**
Display > Variable > Cyclical flags	None Subfunctions: **General configuration**
Display > Variable > Flags	
Display > Variable > Counter	
Display > Variable > Timer	
Display > Energy consumption	None
Display > Windows > NAVIGATOR	None
Display > Windows > PROGRAM	None
Display > Windows > EDITOR	None

15.1.3.1 Variable display

Operator control element	Function group
Refresh	None
Set value	**General configuration**

15.1.3.2 Variable overview

Operator control element	Function group
Display	None
Configure	**General configuration**
Refresh	None
Cancel info	None
Start info	None
Insert	**General configuration**
Delete	**General configuration**
Edit	**General configuration**
OK	**General configuration**
Cancel	None

15.1.4 Menu item Diagnosis

Menu sequence	Function group
Diagnosis > Trace > Configuration Subfunctions: (>>> *15.1.4.1 "Trace" Page 612*)	**Diagnostic functions**
Diagnosis > Trace > Oscilloscope	**Diagnostic functions**
Diagnosis > Logbook > Display Subfunctions: (>>> *15.1.4.2 "TraceLogbook" Page 612*)	**Diagnostic functions**
Diagnosis > Logbook > Configuration	**General configuration**

Menu sequence	Function group
Diagnosis > **Caller stack** Subfunctions: No function group	**Diagnostic functions**
Diagnosis > **Interrupts** Subfunctions: No function group	**Diagnostic functions**
Diagnosis > **Diagnostic monitor**	**Diagnostic functions**
Diagnosis > **KrcDiag**	**Diagnostic functions**

15.1.4.1 Trace

Operator control element	Function group
Start trace	**Diagnostic functions**
Stop Trace	**Diagnostic functions**
Trigger	**Diagnostic functions**
All other operator control elements	**General configuration**

15.1.4.2 TraceLogbook

Operator control element	Function group
Export	**Archive to local HDD/SSD**
Refresh	None
OK	**General configuration**
Save	**General configuration**

15.1.5 Menu item Start-up

Menu sequence	Function group
Start-up > **Start-up wizard**	**General configuration**
Start-up > **Supplementary load data**	**Calibration**
Start-up > **Tool/base management** Subfunctions: **Calibration**	**Calibration**
Start-up > **Calibrate** > **Linear unit**	**Calibration**
Start-up > **Calibrate** > **Linear unit (numeric)**	**Calibration**
Start-up > **Calibrate** > **Tolerances**	**Critical configurations**
Start-up > **Master** > **Dial** Subfunctions: **Mastering**	**Dial mastering**
Start-up > **Master** > **EMD** > **Standard** > **Set mastering**	**Mastering**
Start-up > **Master** > **EMD** > **Standard** > **Check mastering**	**Mastering**
Start-up > **Master** > **EMD** > **With load correction** > **First mastering**	**Mastering**
Start-up > **Master** > **EMD** > **With load correction** > **Teach offset**	**Mastering**
Start-up > **Master** > **EMD** > **With load correction** > **Load mastering** > **With offset**	**Mastering**

Menu sequence	Function group
Start-up > **Master** > **EMD** > **With load correction** > **Load mastering** > **Without offset**	**Mastering**
Start-up > **Master** > **Reference**	**Mastering**
Start-up > **Master** > **Unmaster**	**Mastering**
Start-up > **Software update** > **Automatic**	**General configuration**
Start-up > **Software update** > **Net**	**General configuration**
Start-up > **Service** > **Software limit switch** Subfunctions: **General configuration**	None
Start-up > **Service** > **Long texts**	**General configuration**
Start-up > **Service** > **Maintenance handbook** Subfunctions: **General configuration**	None
Start-up > **Service** > **Start-up mode**	**Start-up mode**
Start-up > **Service** > **Reset safety I/O error**	None
Start-up > **Service** > **Replacement of the wrist** Subfunctions: **Mastering**	None
Start-up > **Service** > **Minimize HMI**	**Critical configurations**
Start-up > **Robot data** Subfunctions: (>>> *15.1.5.1 "Robot data" Page 613*)	None
Start-up > **Network configuration** Subfunctions: (>>> *15.1.5.2 "Network configuration" Page 614*)	None
Start-up > **Additional software** Subfunctions: (>>> *15.1.5.3 "Additional software" Page 614*)	**General configuration**

Menu sequence	User group
Start-up > **Rights management** Subfunctions: **Administrator** **Note:** The rights management subfunctions are not assigned to any function group. They can only be used by the user group **Administrator**.	None

15.1.5.1 Robot data

Button	Function group
Import PID»RDC	**Critical configurations**
Transfer MAM»RDC	**Critical configurations**
Transfer CAL»RDC	**Critical configurations**
Save RDC data	**Archive to local HDD/SSD**

Box	Function group
Controller name	**Critical configurations**
Network archive path	
Domain\User	
User password	
Incorporate controller name into archive name	

15.1.5.2 Network configuration

Element	Function group
“**Internal subnets**” tab	**Critical configurations**
[All elements outside of the tab **Internal subnets**]	**General configuration**

15.1.5.3 Additional software

Operator control element	Function group
New software	**General configuration**
Uninstall	**General configuration**
Restart	**General configuration**
Back	None
Refresh	None
Configure	**General configuration**
Delete path	**General configuration**
Path selection	**General configuration**
Save	**General configuration**
Cancel	None

15.1.6 Menu item Shutdown

Menu item	Function group
Shutdown	None

Subfunctions:

Element	Function group
Start type	**Critical configurations**
Power-off wait time [s]	**Critical configurations**
Power-fail wait time [s]	**Critical configurations**
Force cold start	**General configuration**
Reload files	**Critical configurations**
Power-off delay time (check box)	**Critical configurations**
Power-fail wait time (check box)	**Critical configurations**
Shut down control PC	**General configuration**
Reboot control PC	**General configuration**
Drive bus	**General configuration**

15.1.7 Menu item Help

Menu sequence	Function group
Help > **Info** Subfunctions: **Archive to local HDD/SSD**	None
Help > **Messages** > **System Software**	None
Help > **Documentation** > **System Software**	None

15.1.8 Navigator

Button	Function group
New	**File operations**
Select	**Program selection and deselection**
Cancel	**Program selection and deselection**
Duplicate	**File operations**
Archive > **USB (KCP)**	**Archive to USB drives** & **Partial archiving**
Archive > **USB (cabinet)**	
Archive > **Network**	
Delete	**File operations**
Open	None
Edit	(>>> *15.1.9 "Navigator: Menu Edit" Page 616*)
Error list Subfunctions: No function group	None
Data list	None
Restore > **USB (KCP)**	**Partial restoration**
Restore > **USB (cabinet)**	
Restore > **Network**	
Restore all > **USB (KCP)**	**Restore**
Restore all > **USB (cabinet)**	
Restore all > **Network**	
Save all > **USB (KCP)**	**Archive to USB drives**
Save all > **USB (cabinet)**	
Save all > **Network**	**Archive to network**
EDITOR	None

15.1.9 Navigator: Menu Edit

Button	Function group
New	**File operations**
Open > File/Directory	None
Open > Data list	
Open > Error list	
Mark all	**File operations**
Cut	**File operations**
Copy	**File operations**
Paste	**File operations**
Delete	**File operations**
Duplicate	**File operations**
Archive > USB (KCP)	**Archive to USB drives** & **Partial archiving**
Archive > USB (cabinet)	
Archive > Network	**Archive to network** & **Partial archiving**
Print	**Print functions**
Rename	**File operations**
Properties	**Critical KRL program changes**
Filter	**General configuration**
Select > Without parameters	**Program selection and deselection**
Select > With parameters	**General configuration**
Cancel program	**Program selection and deselection**
Reset program	**Block selection**

15.1.10 Editor: button bar

Button	Function group
Change	**General KRL program changes**
Commands Subfunctions: (>>> *15.1.15 "Editor: Commands menu" Page 619*)	**General KRL program changes**
Motion Subfunctions: **General KRL program changes**	**General KRL program changes**
Open/close fold	UserFolds: None ExpertFolds: **Critical KRL program changes**
Block selection	**Block selection**
TouchUp	**Teach local points**

Button	Function group
Edit Subfunctions: (>>> *15.1.14 "Editor: Menu Edit" Page 618*)	None
Last command	**General KRL program changes**

15.1.11 “Project management” window

Button	Function group
Factory settings	**Critical configurations**
Copy	None
Set as base project	**General configuration**
Save current state	None

Button bar:

Button	Function group
Activate	**General configuration**
Pin	**General configuration**
Unpin	
Copy	None
Delete	**General configuration**
Edit	**General configuration**
Refresh	None

15.1.12 Jog options window: Tabs

General

Element	Function group
Program override	**Program execution settings**
Jog override	**General jog settings**
Program run mode	
Go	**Program execution settings**
Motion	
Single Step	**Critical jog settings**

Keys

Element	Function group
Incremental jogging	**General jog settings**
Key groups	**General jog settings**
Coordinate system	**General jog settings**

Mouse

Element	Function group
Mouse settings	**General configuration**
Coordinate system	**General jog settings**

KCP pos.

Element	Function group
Rotary selector	**General jog settings**

Cur. tool/base

Element	Function group
Tool selection	**General jog settings**
Base selection	
IpoMode selection	

Collision detection

Element	Function group
Default value offset	**General jog settings**
Override collision detection	**Critical jog settings**

15.1.13 Wait messages

Button	Function group
Simulate	**Block selection**

15.1.14 Editor: Menu Edit

Menu item	Function group
Open/close current FOLD	None
Open all FOLDs	
Close all FOLDs	
Clean data list	**Critical KRL program changes**
Cut	**General KRL program changes**
Copy	**General KRL program changes**
Paste	**General KRL program changes**
Delete	**General KRL program changes**
Print	**Print functions**
Search	None
Replace	**Critical KRL program changes**

Menu item	Function group
Mark region	**General KRL program changes**
Marked region > Cart. distance	
Marked region > Transform - Cartesian base	
Marked region > Transform - Cartesian tool	
Marked region > Transform - Cartesian World	
Marked region > Transform - Axis-specific	
Marked region >	
View > DEF line	**Critical KRL program changes**
View > Detail view (ASCII)	**Critical KRL program changes**
View > Line break	None
View > Line spacing normal/small	None
Cancel program	**Program selection and deselection**
Reset program	**Block selection**
Navigator	None

15.1.15 Editor: Commands menu

Button	Function group
Last command	None
Motion > SPTP	**New motion range inline forms**
Motion > SLIN	
Motion > SCIRC	
Motion > SPLINE block	
Motion > SPL	
Motion > PTP SPLINE block	
Motion > PTP	**Old motion range inline forms**
Motion > LIN	
Motion > CIRC	
Motion parameters > Collision detection	None
Logic > WAIT	None
Logic > WAITFOR	
Logic > OUT > OUT	
Logic > OUT > PULSE	
Logic > OUT > SYN OUT	
Logic > OUT > SYN PULSE	
Logic > Couple/decouple IBUS segment	
Logic > Spline trigger	**New motion range inline forms**
Logic > Spline Stop Condition	
Analog output > Static	None
Analog output > Dynamic	None
Comment > Normal	None

Button	Function group
Comment > **Stamp**	None

15.1.16 smartHMI: status bar

Overview

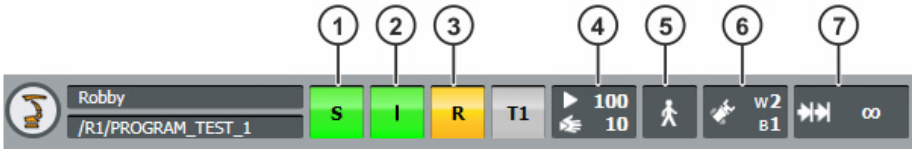


Fig. 15-1: Indicators in the status bar relating to function groups

Item	Designation
1	**Submit interpreter**
2	**Drives**
3	**Robot interpreter**
4	**Overrides**
5	**Program run mode**
6	**Cur. tool/base**
7	**Incremental jogging**

Submit interpreter

Overview of All SUBMIT interpreters	Function group
Select/Start	**General configuration**
Stop	
Deselect	
Display/Assign	

Drives

“Motion conditions” window	Function group
Drives switch	**Program selection and deselection**

Robot interpreter

Program menu	Function group
Cancel program	**Program selection and deselection**
Reset program	**Block selection**

Overrides

“Overrides” window	Function group
Program override	**Program execution settings**
Jog override	**General jog settings**

Program run mode

“Program run mode” window	Function group
Go	**Program execution settings**
Motion	
Single Step	**Critical jog settings**

Cur. tool/base

“Cur. tool/base” window	Function group
Tool selection	**General jog settings**
Base selection	
IpoMode selection	

Incremental jogging

“Incremental jogging” window	Function group
Select increment size from list	**General jog settings**

15.1.17 smartHMI: left sidebar

Element	Function group
WorkVisual icon > **Open** button Subfunctions: (>>> *15.1.11 ""“Project management” window" Page 617*)	None

15.1.18 smartHMI: right sidebar

Element	Function group
Space Mouse status indicator	**General jog settings**
Space Mouse alignment indicator > **Options** button	**General jog settings**
Jog keys status indicator	**General jog settings**
Operation with jog keys	**Jogging with the jog keys**
Operation with mouse	**Jogging using the 6D mouse**
Program override	**Program execution settings**
Jog override	**General jog settings**

16 KUKA Service

16.1 Requesting support

Introduction

This documentation provides information on operation and operator control, and provides assistance with troubleshooting. For further assistance, please contact your local KUKA subsidiary.

Information

The following information is required for processing a support request:

- Description of the problem, including information about the duration and frequency of the fault
- As comprehensive information as possible about the hardware and software components of the overall system

  The following list gives an indication of the information which is relevant in many cases:

  - Model and serial number of the kinematic system, e.g. the manipulator
  - Model and serial number of the controller
  - Model and serial number of the energy supply system
  - Designation and version of the system software
  - Designations and versions of other software components or modifications
  - Diagnostic package KRCDiag

    Additionally for KUKA Sunrise: existing projects including applications

    For versions of KUKA System Software older than V8: archive of the software (KRCDiag is not yet available here.)
  - Application used
  - External axes used

16.2 KUKA Customer Support

Availability

KUKA Customer Support is available in many countries. Please do not hesitate to contact us if you have any questions.

Argentina

Ruben Costantini S.A. (Agentur)
Luis Angel Huergo 13 20
Parque Industrial
2400 San Francisco (CBA)
Argentina
Tel. +54 3564 421033
Fax +54 3564 428877
ventas@costantini-sa.com

Australia

KUKA Robotics Australia Pty Ltd
45 Fennell Street
Port Melbourne VIC 3207
Australia
Tel. +61 3 9939 9656
info@kuka-robotics.com.au
www.kuka-robotics.com.au

Belgium

KUKA Automatisering + Robots N.V.
Centrum Zuid 1031
3530 Houthalen
Belgium
Tel. +32 11 516160
Fax +32 11 526794
info@kuka.be
www.kuka.be

Brazil

KUKA Roboter do Brasil Ltda.
Travessa Claudio Armando, nº 171
Bloco 5 - Galpões 51/52
Bairro Assunção
CEP 09861-7630 São Bernardo do Campo - SP
Brazil
Tel. +55 11 4942-8299
Fax +55 11 2201-7883
info@kuka-roboter.com.br
www.kuka-roboter.com.br

Chile

Robotec S.A. (Agency)
Santiago de Chile
Chile
Tel. +56 2 331-5951
Fax +56 2 331-5952
robotec@robotec.cl
www.robotec.cl

China

KUKA Robotics China Co., Ltd.
No. 889 Kungang Road
Xiaokunshan Town
Songjiang District
201614 Shanghai
P. R. China
Tel. +86 21 5707 2688
Fax +86 21 5707 2603
info@kuka-robotics.cn
www.kuka-robotics.com

Germany

KUKA Deutschland GmbH
Zugspitzstr. 140
86165 Augsburg
Germany
Tel. +49 821 797-1926
Fax +49 821 797-41 1926
Hotline.robotics.de@kuka.com
www.kuka.com

France

KUKA Automatisme + Robotique SAS
Techvallée
6, Avenue du Parc
91140 Villebon S/Yvette
France
Tel. +33 1 6931660-0
Fax +33 1 6931660-1
commercial@kuka.fr
www.kuka.fr

India

KUKA India Pvt. Ltd.
Office Number-7, German Centre,
Level 12, Building No. - 9B
DLF Cyber City Phase III
122 002 Gurgaon
Haryana
India
Tel. +91 124 4635774
Fax +91 124 4635773
info@kuka.in
www.kuka.in

Italy

KUKA Roboter Italia S.p.A.
Via Pavia 9/a - int.6
10098 Rivoli (TO)
Italy
Tel. +39 011 959-5013
Fax +39 011 959-5141
kuka@kuka.it
www.kuka.it

Japan

KUKA Japan K.K.
YBP Technical Center
134 Godo-cho, Hodogaya-ku
Yokohama, Kanagawa
240 0005
Japan
Tel. +81 45 744 7531
Fax +81 45 744 7541
info@kuka.co.jp

Canada

KUKA Robotics Canada Ltd.
2865 Argentia Road, Unit 4-5
Mississauga
Ontario L5N 8G6
Canada
Tel. +1 905 858-5852
Fax +1 905 858-8581
KUKAFocusCenter@KUKARobotics.com
www.kukarobotics.ca

Korea

KUKA Robotics Korea Co. Ltd.
RIT Center 306, Gyeonggi Technopark
1271-11 Sa 3-dong, Sangnok-gu
Ansan City, Gyeonggi Do
426-901
Korea
Tel. +82 31 501-1451
Fax +82 31 501-1461
info@kukakorea.com

Malaysia

KUKA Robot Automation (M) Sdn Bhd
South East Asia Regional Office
No. 7, Jalan TPP 6/6
Taman Perindustrian Puchong
47100 Puchong
Selangor
Malaysia
Tel. +60 (03) 8063-1792
Fax +60 (03) 8060-7386
info@kuka.com.my

Mexico

KUKA de México S. de R.L. de C.V.
Progreso #8
Col. Centro Industrial Puente de Vigas
Tlalnepantla de Baz
54020 Estado de México
Mexico
Tel. +52 55 5203-8407
Fax +52 55 5203-8148
info@kuka.com.mx
www.kuka-robotics.com/mexico

Norway

KUKA Sveiseanlegg + Roboter
Sentrumsvegen 5
2867 Hov
Norway
Tel. +47 61 18 91 30
Fax +47 61 18 62 00
info@kuka.no

Austria

KUKA CEE GmbH
Gruberstraße 2-4
4020 Linz
Austria
Tel. +43 732 784 752 0
Fax +43 732 793 880
KUKAAustriaOffice@kuka.com
www.kuka.at

Poland

KUKA CEE GmbH Poland
Spółka z ograniczoną odpowiedzialnością
Oddział w Polsce
Ul. Porcelanowa 10
40-246 Katowice
Poland
Tel. +48 327 30 32 13 or -14
Fax +48 327 30 32 26
ServicePL@kuka.com

Portugal

KUKA Robots IBÉRICA, S.A.
Rua do Alto da Guerra n° 50
Armazém 04
2910 011 Setúbal
Portugal
Tel. +351 265 729 780
Fax +351 265 729 782
info.portugal@kukapt.com
www.kuka.com

Russia

KUKA Russia OOO
1-y Nagatinskiy pr-d, 2
117105 Moskau
Russia
Tel. +7 495 665-6241
support.robotics.ru@kuka.com

Sweden

KUKA Svetsanläggningar + Robotar AB
A. Odhners gata 15
421 30 Västra Frölunda
Sweden
Tel. +46 31 7266-200
Fax +46 31 7266-201
info@kuka.se

Switzerland

KUKA Roboter CEE GmbH
Linz, Zweigniederlassung Schweiz
Heinrich Wehrli-Strasse 27
5033 Buchs
Switzerland
Tel. +41 62 837 43 20
info@kuka-roboter.ch

Slovakia

KUKA CEE GmbH
organizačná zložka
Bojnická 3
831 04 Bratislava
Slovakia
Tel. +420 226 212 273
support.robotics.cz@kuka.com

Spain

KUKA Iberia, S.A.U.
Pol. Industrial
Torrent de la Pastera
Carrer del Bages s/n
08800 Vilanova i la Geltrú (Barcelona)
Spain
Tel. +34 93 8142-353
comercial@kukarob.es

South Africa

Jendamark Automation LTD (Agentur)
76a York Road
North End
6000 Port Elizabeth
South Africa
Tel. +27 41 391 4700
Fax +27 41 373 3869
www.jendamark.co.za

Taiwan

KUKA Automation Taiwan Co. Ltd.
1F, No. 298 Yangguang ST.,
Nei Hu Dist., Taipei City, Taiwan 114
Taiwan
Tel. +886 2 8978 1188
Fax +886 2 8797 5118
info@kuka.com.tw

Thailand

KUKA (Thailand) Co. Ltd.
No 22/11-12 H-Cape Biz Sector Onnut
Sukhaphiban 2 road, Prawet
Bangkok 10250
Thailand
Tel. +66 (0) 90-940-8950
HelpdeskTH@kuka.com

Czech Republic

KUKA Roboter CEE GmbH
organizační složka
Pražská 239
25066 Zdiby
Czech Republic
Tel. +420 226 212 273
support.robotics.cz@kuka.com

Hungary

KUKA HUNGÁRIA Kft.
Fö út 140
2335 Taksony
Hungary
Tel. +36 24 501609
Fax +36 24 477031
info@kuka-robotics.hu

USA

KUKA Robotics Corporation
51870 Shelby Parkway
Shelby Township
48315-1787
Michigan
USA
Tel. +1 866 873-5852
Fax +1 866 329-5852
CustomerService@kuka.com
www.kuka.com

United Kingdom

KUKA Robotics UK Ltd
Great Western Street
Wednesbury West Midlands
WS10 7LL
United Kingdom
Tel. +44 121 505 9970
Fax +44 121 505 6589
service@kuka-robotics.co.uk
www.kuka-robotics.co.uk

Index

#BSTEP.....328
#CSTEP.....328
#GO.....327
#IGNORE.....376, 377
#ISTEP.....327
#MSTEP.....327
#PSTEP.....328
$.....450
$ACCU_STATE.....122
$ADAP_ACC.....342, 350
$ADVANCE.....328
$ALARM_STOP.....245, 246
$ALARM_STOP_INTERN.....245, 246
$ANIN.....436
$ANOUT.....436
$AUT.....247
$BRAKE_SIG.....264
$BRAKES_OK.....279
$BRAKETEST_MONTIME.....278
$BRAKETEST_REQ_EX.....278
$BRAKETEST_REQ_INT.....278
$BRAKETEST_WARN.....279
$BRAKETEST_WORK.....279
$BRK_DEL.....560
$BWD_INFO.....341
$BWDSTART.....341
$CHCK_MOVENA.....243
$CIRC_MODE.....379
$CIRC_TYPE.....378
$CMD.....588
$COLL_ALARM.....353
$COLL_ENABLE.....353
$COLLMON_ACTIVE_COM.....351
$COLLMON_ACTIVE_PRO.....351
$COLLMON_MAX.....347, 353
$COLLMON_TOL_ACT.....353
$COLLMON_TOL_COM.....352
$COLLMON_TOL_COM_DEF.....352
$COLLMON_TOL_PRO.....352
$COLLMON_TOL_PRO_DEF.....352
$CONF_MESS.....243
$CONST_VEL.....487
$CONST_VEL_C.....487
$COOLDOWN_TIME.....235
$COULD_START_MOTION.....64
$CRIT_PERI_ACK.....136
$CRIT_PERI_ACK_REQ.....137
$DIST_LAST.....541
$DIST_NEXT.....541
$DRIVES_OFF.....244
$DRIVES_ON.....244
$ECO_LEVEL.....224
$ENERGY_CONFIG_STATE.....118
$ENERGY_INTERIM.....120
$ENERGY_MEASURING.....116
$ENERGY_PERIOD.....119
$ENERGY_TOTAL.....120
$ENERGYMODULE.....117
$ERR.....498, 586
$EX_AX_IGNORE.....488
$EXT.....247
$EXT_START.....243
$HOLDING_TORQUE.....265, 266
$I_O_ACT.....244
$I_O_ACTCONF.....245
$IMPROVED_COLLMON.....342, 350, 351
$IN.....436
$IN_HOME.....246
$INTERPRETER.....586, 587
$LDC_CONFIG.....193
$LDC_LOADED.....192
$LDC_RESULT.....194
$LOAD_BWINI.....341
$MOVE_ENABLE.....243
$NEAR_POSRET.....247
$ON_PATH.....246
$ORI_TYPE.....358, 376
$OUT.....436
$PAL_MODE.....143
$PATHTIME.....484
$PERI_RDY.....64, 245
$POS_ACT.....543
$PRO_ACT.....246
$PRO_IP.....596
$PRO_IP….....586
$PRO_MODE.....328
$PRO_MODE….....586
$PRO_MOVE.....246
$PRO_NAME….....586
$PRO_STATE0.....586
$PROG_INFO.....587
$RC_RDY1.....245
$ROB_CAL.....245
$ROB_STOPPED.....247
$ROBRUNTIME.....111, 113
$SPL_ORI_JOINT_AUTO.....378
$STOP_CONST_VEL_RED.....486
$STOPMESS.....245
$T1.....247
$T2.....247
$TOOL_DIRECTION.....169
$TOOL_DIRECTION_LIN_CIRC.....169
$TORQUE_AXIS_ACT.....263, 266
$TORQUE_AXIS_LIMITS.....264
$TORQUE_AXIS_MAX.....264
$TORQUE_AXIS_MAX_0.....264
$US2_VOLTAGE_ON.....134
$USER_SAF.....65, 245
$VW_BACKWARD.....341
$VW_CYCFLAG.....341
$VW_MOVEMENT.....341
$WAIT_FOR_ON0.....586
$WAIT_FOR0.....586
$WARMUP_CURR_LIMIT.....235
$WARMUP_MIN_FAC.....235
$WARMUP_RED_FAC_ACT.....235
$WARMUP_RED_VEL.....235
$WARMUP_SLEW_RATE.....236
$WARMUP_TIME.....235

_TYP....457
_TYPE....458
2006/42/EC:2006....50
2014/30/EU:2014....50
2014/68/EU:2014....50
3-point (method)....183
6D mouse....57
95/16/EC....50

A

A6, mastering position....160
ABC 2-point (method)....182
ABC world (method)....181
Absolute value (mathematical function)....551
Accessories....19, 23
Activating, project....311
ACTIVE (state)....573
Actual position....97
Addition....542
Addition, geometric....543
Administrator (user group)....77
Advance run....328
Advance run stop....357, 489
Analog inputs....506
Analog outputs....507
ANIN....506
ANOUT....437, 507
ANSI/RIA R.15.06-2012....50
Appendix....607
APPL_RUN....246
Applied norms and regulations....50
Approximate positioning....357, 396
Approximate positioning, homogenous....537
Approximate positioning, mixed....537
Arc cosine (mathematical function)....551
Arc sine....552
Arc tangent (mathematical function)....551
Archiving overview....307
Archiving, logbook....310
Archiving, network....309
Archiving, to USB stick....308
Areas of validity....453
ASYPTP....584
Automatic mode....47
Auxiliary point....356, 461, 465
Axis limitation, mechanical....36
Axis monitoring functions, configuring....209
Axis range....25
Axis, active....273
Axis, requested....273

B

Backup configuration (window)....321
Backup Manager....316
Backup Manager, configuring....321
Backup, option packages....316, 317
Backup, projects....316, 317
Backup, RDC data....316, 317
Backward motion (using “Start backwards” key)....336
Backward motion (using jog keys)....92
Backward motion, configuring....237
Backward, preventing start....553
BACKWARD_STEP....237
BASE....172
Base calibration....172
BASE coordinate system....79, 172
Battery state....122
BCO run....333
Bit operators....548
Block coincidence....333
Block pointer....298, 329
Block selection....334, 364
BRAKE....521, 526, 527
Brake check, automatic....273, 287
Brake defect....39
Brake delay....560
BRAKE F....527
BRAKE FF....527
Brake release device....37
Brake test....272
Brake test, cycle time....273
Brake test, manual....285
Brake test, programs....274
Brake test, signals....277, 279
Brake test, teaching positions....281
Brake, defective....281, 282, 285
BrakeTestAxes.src....274
BrakeTestBack.src....275, 282
BrakeTestPark.src....274, 282
BrakeTestReq.src....274
BrakeTestStart.src....275, 281
Braking distance....25
Branch, conditional....493
BSTEP....328
Bypassing workspace monitoring....96

C

Calibration....169
Calibration tolerances, defining....236
Calibration, base....172
Calibration, linear unit....188
Call by Reference....516
Call by Value....516
Caller stack....596
Caller stack (menu item)....596
CANCEL (command)....589
Cancel, program....297
CASE....502
CAST_FROM....542
CAST_TO....542
CCLOSE....542
CE mark....24
CELL.SRC....335
CHANNEL....542
Checksum , safety configuration....216
CIOCTL....542

CIRC.....461
CIRC motion.....394
CIRC, motion type.....356
CIRC_REL.....465
Circular angle.....383
Circular motion.....461, 465
CK, monitoring.....35
Cleaning work.....47
Close all FOLDs (menu item).....303
Cold start.....71, 575, 580
Cold start, initial.....68, 70, 71, 603
CollDetect_UserAction.....342
Collision detection.....342, 348
Collision detection, configuring.....345, 347
Collision detection, system variables.....350
Command interface.....588
Comment.....304
Communication.....585
Comparison, data from kernel system and hard drive.....272
Conditional branch.....493
Configuration.....201
Configuration (menu item).....217
Configuring CELL.SRC.....239
Connecting cables.....19, 23
Connection manager.....54, 56
CONST.....455
CONST_VEL.....485
Constant velocity range.....411, 423, 485
Constants.....454, 455
CONTINUE.....489
Continuous Path.....355
Controller name, display.....63
Coordinate system for jog keys.....62
Coordinate system for Space Mouse.....61
Coordinate systems.....79
Coordinate systems, angles.....80
Coordinate systems, orientation.....80
COPEN.....542
Copy.....306
Cosine (mathematical function).....551
Counterbalancing system.....48
Counters, displaying.....109
CP motions.....355
CP spline block.....398, 469
CREAD.....542
Creating a new folder.....291
Creating a new program.....291
Cut.....306
CWRITE.....542, 588

D

Danger zone.....25
DAT.....450
Data list.....450
Data type, user-defined.....454, 457, 458
Data types.....451
Data, restoring.....310
DECL.....455
Declaration of conformity.....24
Declaration of incorporation.....23, 24
Decommissioning.....48
DEF line (menu item).....301
DEF line, displaying/hiding.....301
DEFAULT.....503
DEFFCT ... ENDFCT.....514
Delay time, power-off.....69, 71
Delay time, power failure.....70
DELETE_BACKWARD_BUFFER.....553
Deselecting, submit interpreters.....576, 577
Detail view (ASCII) (menu item).....302
Detail view, activating.....301
Diagnosis.....593
Diagnostic monitor (menu item).....599
Dial gauge.....157
Directory structure.....292
Display (menu item).....105
Displaying a variable, single.....103, 105
Displaying the logbook.....593
Displaying variables, in overview.....105
Displaying, robot controller information.....111
Displaying, robot information.....111
Disposal.....48
DISTANCE.....529
Division.....542
Documentation, industrial robot.....17
Drive bus.....70
Drives, switching on/off.....65

E

EC declaration of conformity.....24
Edit (button).....62
Editor.....297
Electromagnetic compatibility (EMC).....51
Electromagnetic compatibility (EMC):.....51
ELSE.....493
EMC Directive.....24, 50
EMERGENCY STOP.....54, 57
EMERGENCY STOP device.....32, 33, 38
EMERGENCY STOP, external.....33, 42
EMERGENCY STOP, local.....42
EN 60204-1:2006/A1:2009.....51
EN 61000-6-2:2005.....51
EN 61000-6-4:2007 + A1:2011.....51
EN 614-1:2006+A1:2009.....50
EN ISO 10218-1:2011.....50
EN ISO 12100:2010.....50
EN ISO 13849-1:2015.....50
EN ISO 13849-2:2012.....50
EN ISO 13850:2015.....50
Enabling device.....33, 38
Enabling device, external.....34
Enabling switches.....33
End user license agreement.....112
ENDFCT.....514
ENDFOR.....490
ENDIF.....493
Endless loop.....493

ENDLOOP..... 493
ENDSPLINE..... 469, 470
ENDSWITCH..... 502
ENDWHILE..... 505
Energy consumption, measuring..... 114
ENUM..... 457
Enumeration type..... 457
ERR_CLEAR()..... 495
ERR_RAISE()..... 495
EtherNet/IP interfaceWindows interface..... 203
EULA..... 112
Even parity..... 244
EXIT..... 490, 493
Expert (user group)..... 77
Expert Submit (template)..... 582
Export (button)..... 217
Extended submit..... 573
Extended submit interpreter..... 573
External axes..... 23, 111
External axis..... 27

F

FALSE..... 513
Faults..... 40
Field bus interface..... 203
Field bus, Ethernet-based..... 201
File list..... 292
File, properties..... 293
Filter..... 293
Find..... 306
Firmware, updating..... 605
First mastering..... 151, 161
Flags, displaying..... 107, 108
FLANGE coordinate system..... 80, 171
Folder, creating..... 291
Folder, properties..... 293
Folds..... 303
Folds, creating..... 305
Folds, displaying..... 302
Fonts..... 449
FOR..... 504
FOR ... TO ... ENDFOR..... 490
FORWARD()..... 553
Frame operation..... 543
FREE (state)..... 574
FREE_PRESELECTED (state)..... 574
FREE_UNSELECTED (state)..... 574
Function groups..... 219, 221
Function test..... 41
Function, calling..... 514
Function, syntax..... 514

G

General safety measures..... 38
Geometric addition..... 543
GET_AXESMASK()..... 288
GET_BRAKETEST_TIME()..... 289
Global..... 453
GLOBAL (interrupt declaration)..... 521
Global, point..... 432
Global_Points.dat..... 433
GO (program run mode)..... 327
GOTO..... 491

H

HALT..... 492
Hardware, options..... 133
Hardware, updating firmware..... 605
Hazardous substances..... 48
Header..... 292
Help, messages..... 72
Hibernate..... 71
HOME position..... 300
Homogenous approximate positioning..... 537

I

I/O driver, reconfiguring..... 208
I/Os, reconfiguring..... 208
Identification plate..... 55, 59
IF ... THEN ... ENDIF..... 493
IN parameters..... 516
Increment..... 92
Incremental jogging..... 91
Indirect (method)..... 185
Industrial robot..... 19, 23
Info (menu item)..... 111
Inline form, names..... 391
Inputs/outputs..... 585
Inputs/outputs, analog..... 101, 435
Inputs/outputs, Automatic External..... 102, 240
Inputs/outputs, digital..... 100, 435
Installation..... 601
Intended use..... 20, 23
INTERIMENERGY()..... 121
INTERN.ZIP..... 308, 309
Interrupt..... 520
Interrupt declaration..... 520
INTERRUPT DISABLE..... 524
INTERRUPT ENABLE..... 524
INTERRUPT OFF..... 523
INTERRUPT ON..... 523
Interrupt program..... 520
Interrupts (menu item)..... 597
Interrupts, displaying..... 597
Introduction..... 17
INV_POS()..... 554
INVERSE..... 556
IP addresses..... 201
ISTEP..... 327
IT security..... 40

J

Jerk..... 401, 402, 407, 409, 415, 417
Jog keys..... 54, 57, 81, 87, 88
Jog mode..... 36, 38

Jog override....86
Jogging, axis-specific....80, 87
Jogging, Cartesian....80, 88, 91
Jogging, external axes....95
Jogging, robot....80
Jump....491

K

Keyboard....54, 57
Keyboard key....54, 57
Keypad....62
Keywords....450
Kinematic system, external....174
Kinematics group....62, 82
KLI, configuring....201
KRCDiag....599
KRL instructions, for SUB programs....584
KUKA Customer Support....111, 623
KUKA Line Interface, configuring....201
KUKA Service....623
KUKA smartHMI....61
KUKA smartPAD....25, 53
KUKA smartPAD-2....25

L

Labeling....37
Language....72
Liability....23
Licenses....18
LIN....460
LIN motion....393
LIN, motion type....356
LIN_REL....463
Line break (menu item)....302
Line mark for mastering....161
Linear motion....460, 463
Linear unit....23, 186
Load data....190
Load data verification....192
Logbook....593
Logbook, configuring....595
Long texts, exporting....194
Long texts, importing....194
LOOP ... ENDLOOP....493
Loss of mastering....151, 155, 160, 165
Low Voltage Directive....24

M

Machine data....42, 111, 113
Machine data, configuration....132
Machinery Directive....24, 50
Main menu key....57
Main menu, calling....66
Maintenance....47, 198
MAMES....196
Manipulator....19, 23, 25
Manual mode....45
Mastering....144
Mastering after maintenance work....159
Mastering marks....147
Mastering methods....145
Mastering position, A6....160
Mastering, deleting....166
Mechanical end stops....36
MEMD....146, 160
Message help....72
Message window....61
Messages, displaying help....73
Micro Electronic Mastering Device....146, 160
Minimizing KUKA smartHMI....66
Mixed approximate positioning....537
Mode selector switch....54, 56
Modifying a logic instruction....448
Modifying a variable....103
Modifying coordinates....425
Modifying motion parameters....425
Modifying variables....105
Module....450
Monitoring functions, checking....214
Monitoring time....273
Monitoring, physical safeguards....31
Monitoring, velocity....35
Motion conditions (window)....64
Motion programming, basic principles....355
Motion types....355
Motor, exchange....159
MSTEP....327
Multi Submit....586
Multiplication....542

N

Name, archive....113
Name, control PC....112
Name, controller....112, 113
Names....450
Names, in inline form....391
Navigator....292
Non-rejecting loop....502
Numeric entry, linear unit....189

O

Odd parity....244
Offset....151, 154, 160, 164, 438
ON_ERROR_PROCEED....494
Online documentation....72
Online optimizing....272
Open all FOLDs (menu item)....303
Open source....18
Opening a program....297
Operating hours....113
Operating hours meter....113
Operating mode after reconfiguration....208
Operating mode after start....68
Operating mode, changing....78
Operation....53

Operator (user group)..... 78
Operator safety..... 29, 31, 38, 65
Operator safety acknowledgement..... 134
Operator, geometric.....543
Operators.....28
Operators for bit operations.....548
Operators for comparison operations.....546
Operators, arithmetic.....542
Operators, logic.....547
Operators, priority.....550
Option packages, backup.....316, 317
Option packages, display..... 112
Option packages, restoring.....316
Options..... 19, 23
Orientation behavior, SCIRC.....379
Orientation control, LIN, CIRC.....358
Orientation control, spline..... 376
OUT..... 436
OUT parameters.....516
Output, analog.....437
Output, digital..... 436
Overload..... 39
Override.....86, 331
Override (menu item).....97
Overriding, power failure.....70
Overview of the industrial robot.....19

P

Palletizing robot.....179
Palletizing robots.....143, 171
Panic position.....33
Parameters, transferring.....516
Parity.....244
Parity bit.....244
Parity, even.....244
Parity, odd.....244
Password, changing.....219
Paste.....306
PATH.....533
Performance Level..... 29
Peripheral contactor..... 45, 134, 142
Personnel.....27
PGNO_FBIT.....242
PGNO_FBIT_REFL..... 245
PGNO_LENGTH.....242
PGNO_PARITY.....242
PGNO_REQ.....246
PGNO_TYPE.....241
PGNO_VALID..... 242
Pinning..... 311
Plant integrator.....27
PLC_ROB_STOP().....559
PLC_ROB_STOP_RELEASE()..... 559
Point-to-point.....355
Point-to-point motion.....459, 462
Point correction, defining limits.....231
Point, global.....432
Positionally accurate robot, checking activation.....142
Positioner.....23
Power-off, delay time.....69, 71
Power failure, delay time..... 70
Power failure, overriding..... 70
Pre-mastering position..... 147, 148
Pressure Equipment Directive.....48, 50
Preventive maintenance work.....47
Printing, program.....307
Priority.....521, 530, 534
Probe..... 146
Product description.....19
PROFIenergy..... 114
PROFINET interface.....203
Program execution..... 327
Program execution control.....489
Program lines, deleting..... 305
Program override.....331
Program run mode, selecting..... 327
Program run modes..... 327
Program, canceling.....297
Program, closing.....298
Program, creating.....291
Program, editing..... 304
Program, opening.....297
Program, printing.....307
Program, selecting.....297
Program, starting.....332, 333
Program, stopping.....333, 335
Project management (window).....313
Project, activating..... 311
Project, inactive.....314
Projects, backup.....316, 317
Projects, restoring.....316
Properties, file or folder..... 293
Protective equipment.....36
PTP.....459
PTP motion.....391
PTP spline block.....398, 470
PTP, motion type.....355
PTP_REL.....462
PTP_SPLINE ... ENDSPLINE.....470
PUBLIC.....454
Pulse.....437, 508
Pulse, path-related..... 446

R

RDC data, restoring..... 320
RDC, data backup..... 114, 316, 317
RDC, exchange..... 159
Re-teaching.....425
Reaction distance.....25
Recommissioning..... 41, 125
Reference mastering..... 159
REFLECT_PROG_NR.....242
Rejecting loop.....505
Release device.....37
Renaming a file..... 291
Renaming a folder.....291
Repair..... 47

REPEAT ... UNTIL....502
Replace....306
RESET (command).... 589
RESET (state).... 574
RESET_TORQUE_LIMITS.... 584
Resetting a program....335
Resetting, submit interpreters.... 577
Restoration, option packages.... 316
Restoration, projects....316
Restoring RDC data....320
RESUME....527
Reteaching, defining limits.... 231
RETURN.... 515
Rights management.... 220
ROB_STOP().... 558
ROB_STOP_RELEASE().... 558
Robot controller....19, 23
Robot controller, switching on....125
Robot data (menu item).... 112
Robot interpreter....573
ROBROOT coordinate system....79
RUN (command)....589
Runtime variable....453

S

Safe operational stop....25, 35
Safeguards, external.... 37
Safety....23
Safety configuration, Checksum.... 216
Safety configuration, export.... 217
Safety configuration, import.... 217
Safety controller....30
Safety functions....29, 38
Safety functions, overview.... 29
Safety instructions.... 17
Safety maintenance (user group).... 77
Safety of machinery....50, 51
Safety options....26
Safety recovery (user group).... 77
Safety STOP 0.... 25
Safety STOP 1.... 25
Safety STOP 2.... 26
Safety STOP 0.... 25
Safety STOP 1.... 25
Safety STOP 2.... 26
Safety stop, external.... 35
Safety zone.... 25, 28
Safety, general....23
SCIRC....472
SCIRC motion, programming....415
SCIRC segment, programming....404
SCIRC_REL....477
Screenshot, smartPAD.... 72
SCTLCRC.XML....216
SEC....504
Selected section (marked region)....307
Selecting a program....297
Selecting the base....87
Selecting the operating mode.... 29, 30
Selecting the tool....87
Selecting, range in program.... 305
SEMD.... 146, 151
Serial number.... 113
Service life....25, 111
SET_BRAKE_DELAY()....560
SET_TORQUE_LIMITS.... 258, 261, 584
Shutdown (menu item)....68
Shutting down the robot controller.... 68
SIGNAL....512
Signal diagrams....249
Signals, brake test....277, 279
Simulation.... 46
Sine (mathematical function)....551
Single (menu item).... 103, 105
Single point of control.... 48
Single Submit.... 586
Singularities.... 388
Singularity, CP spline....376, 378
Singularity, LIN/CIRC....358
SLIN....471
SLIN motion, programming....413
SLIN segment, programming....402
SLIN_REL....475
smartHMI....19, 61
smartPAD....26, 39, 53
Soft axes....255
Software.... 19, 23
Software limit switches....36, 38, 167
Software limit switches, modifying....167
Space Mouse.... 54, 57, 81, 88, 90, 91
Special characters....391
SPL.... 473
SPL segment, programming.... 402
SPL_REL.... 478
SPLINE ... ENDSPLINE....469
Spline block, programming....398
Spline segment....361
Spline, motion type....361
SPOC....48
SPS.SUB....573, 582
sps.sub, editing....582
SPTP....474
SPTP motion, programming....418
SPTP segment, programming....405
SPTP_REL....479
Square root (mathematical function).... 551
SRC.... 450
SREAD....542
Stamp....304
Standard Electronic Mastering Device 146, 151
Standstill monitoring....210
Start-up....41, 125
Start-up mode....44
Start-up wizard.... 132
Start backwards key.... 54, 57
Start key....54, 55, 57, 58
Start type, KSS....67
Start types.... 71
Starting a program, automatic.... 333

Starting a program, manual.....332
Starting Automatic External mode.....335
Starting the KSS.....66
Starting, submit interpreter.....578
Starting, submit interpreters.....575, 577, 578
Starting, system submit interpreter.....578
States, submit interpreter.....573
Status.....384, 385
Status bar.....61, 63, 292, 574
Status indicator.....574
Status keys.....54, 57
STEP.....491
STOP (command).....589
STOP (state).....573
STOP 0.....24, 26
STOP 1.....24, 26
STOP 2.....24, 26
Stop category 0.....26
Stop category 1.....26
Stop category 2.....26
Stop category 1, Drive Ramp Stop.....26
STOP key.....54, 57
Stop reactions.....28
STOP WHEN PATH.....487
STOP 1 - DRS.....26
Stopping a program.....333, 335
Stopping distance.....25, 28
Stopping, robot.....526, 558
Stopping, submit interpreters.....576, 577
Storage.....48
Storage capacities.....112
String variable length after initialization.....567
String variable length in the declaration.....566
String variable, deleting contents.....567
String variables.....564
String variables, comparing contents.....569
String variables, copying.....570
String variables, extending.....568
String variables, searching.....569
STRUC.....458
Structure type.....458
SUB program, creating.....581
Submit (template).....582
Submit interpreter.....63, 573
Subprogram, calling.....513
Subprograms, of a SUB program.....584
Subtraction.....542
Supplementary load data (menu item).....192
Support request.....623
SWITCH ... CASE ... ENDSWITCH.....502
Switching action, path-related.....441
Switching on the robot controller.....66
SWRITE.....542
Symbols.....449
SYN OUT.....441
SYN PULSE.....446
System integrator.....24, 27
System requirements.....20, 601
System submit.....573
System submit interpreter.....573
System variables.....341
System variables, submit interpreter.....585

T

T1 (operating mode).....27
T2 (operating mode).....27
Tangent (mathematical function).....551
TCP.....170
Teach pendant.....19, 23
Teaching.....425
Technology packages.....19, 391, 450
Technology packages, display.....112
Terms used, safety.....24
Time block.....482
TIME_BLOCK.....481
Timers, displaying.....110
TOOL.....170
Tool Center Point.....170
TOOL coordinate system.....79, 170
Tool direction.....169
TORQMON.....342
Torque mode, diagnosis.....263
Torque mode, examples.....257, 266
Torque mode, overview.....255
Touch screen.....53, 62
Trace (jogging option).....83
Trademarks.....18
Training.....17
Transforming coordinates.....426
Transportation.....40
TRIGGER.....529, 533
Trigger, for spline inline form.....409
Turn.....384, 388
Turn-tilt table.....23
Type, robot.....111
Type, robot controller.....111

U

Unmastering.....166
UNTIL.....502
Update.....603
Update, firmware of hardware.....605
US2.....45, 134, 142
USB sticks.....20
Use, contrary to intended use.....23
Use, improper.....23
User.....25, 27
User (user group).....77
User group, changing.....76
User groups.....77
User interface.....61
User rights.....219

V

Variable correction.....103
Variable overview, configuring.....217
VARSTATE().....105, 563

VECTORMOVEOFF..... 584
VECTORMOVEON..... 584
Velocity..... 86, 331
Velocity monitoring..... 35
Version, kernel system..... 111
Version, operating system..... 112
Version, robot controller.....111
Version, user interface.....111
Voltage.....438, 439

W

WAIT.....439, 504
WAIT FOR..... 504
Wait function, signal-dependent.....440
WAIT SEC..... 504
Wait time..... 439, 504
WAITFOR.....440
Warm-up..... 233
Warnings..... 17
WHILE ... ENDWHILE.....505
Windows interface.....66, 201
WITH (permissible system variables..... 480
Workspace.....25, 28
Workspaces, axis-specific..... 225
Workspaces, bypassing monitoring..... 96
Workspaces, Cartesian..... 225
Workspaces, mode..... 228, 231
WORLD coordinate system.....79
Wrist root point.....228

X

XML export..... 217
XML import..... 217
XYZ (method)..... 180
XYZ 4-point (method).....179
XYZ Reference (method)..... 180