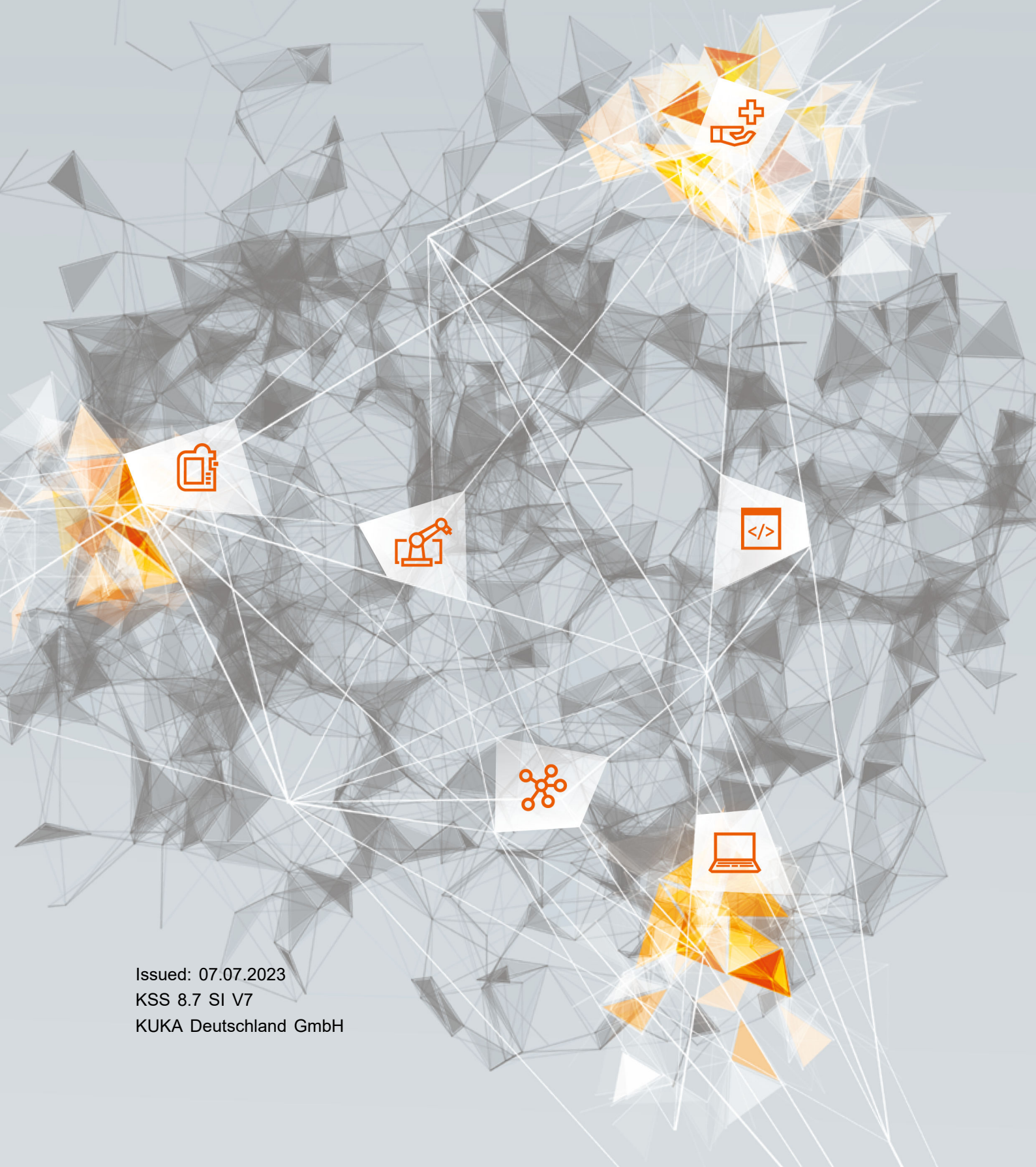
KUKA System Software

KUKA System Software 8.7

Operating and Programming Instructions for System Integrators



Issued: 07.07.2023
KSS 8.7 SI V7
KUKA Deutschland GmbH

© Copyright 2023
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 8.7 SI (PDF) en
	PB14656
Book structure:	KSS 8.7 SI V7.2
	BS13208
Version:	KSS 8.7 SI V7

Contents

1	**Introduction**	**21**
1.1	Target group	21
1.2	Industrial robot documentation	21
1.3	Representation of warnings and notes	21
1.4	Trademarks	22
1.5	Licenses	22
2	**Product description**	**23**
2.1	Product description	23
2.2	Overview of the industrial robot	23
2.3	Network connections	24
2.4	System requirements	25
2.5	KUKA USB sticks	25
2.6	Intended use and misuse	25
3	**Safety**	**27**
3.1	General	27
3.1.1	Disclaimer	27
3.1.2	EC declaration of conformity and declaration of incorporation	27
3.1.3	Terms in the “Safety” chapter	28
3.2	Personnel	31
3.3	Workspace, safety zone and danger zone	32
3.3.1	Determining stopping distances	33
3.4	Triggers for stop reactions: KSS	33
3.5	Triggers for stop reactions: VSS	34
3.6	Safety functions	34
3.6.1	Overview of the safety functions	34
3.6.2	Safety controller	35
3.6.3	“Operator safety” signal: KSS	36
3.6.4	“Operator safety” signal: VSS	36
3.6.5	EMERGENCY STOP devices	37
3.6.6	Logging off from the higher-level safety controller	38
3.6.7	Enabling device	38
3.6.8	External enabling device	40
3.6.9	Velocity monitoring in T1	40
3.7	Additional protective equipment	40
3.7.1	Jog mode	40
3.7.2	Software limit switches	40
3.7.3	Mechanical end stops	41
3.7.4	Mechanical axis limitation (optional)	41
3.7.5	Options for moving the manipulator without drive energy	41
3.7.6	Labeling on the industrial robot	42
3.7.7	External safeguards	42
3.7.8	“Drives ready” lamp	43
3.8	Operating mode selection: KSS	43
3.9	Overview of operating modes and safety functions: KSS	44
3.10	Operating mode selection: VSS	45

3.11	Overview of operating modes and safety functions: VSS	46
3.12	Safety measures	46
3.12.1	General safety measures	46
3.12.2	IT security	48
3.12.3	Transportation	49
3.12.4	Start-up and recommissioning: KSS/VSS	49
3.12.4.1	Checking machine data and safety configuration	51
3.12.4.2	Start-up mode	54
3.12.5	Manual mode	55
3.12.6	Simulation	56
3.12.7	Automatic mode	56
3.12.8	Maintenance and repair	56
3.12.9	Decommissioning, storage and disposal	58
3.12.10	Safety measures for single point of control	58
4	**Operation**	**61**
4.1	smartPAD	61
4.1.1	Front of smartPAD-2	61
4.1.2	Rear of smartPAD-2	63
4.1.3	Disconnecting and connecting the smartPAD	64
4.2	KUKA smartHMI user interface	65
4.2.1	Keypad	67
4.2.2	Status bar	67
4.2.3	Drives status indicator and Motion conditions window	68
4.2.4	Minimizing KUKA smartHMI (displaying Windows interface)	70
4.3	Switching on the robot controller and starting the KSS	70
4.4	Calling the main menu	71
4.5	Defining the start type for KSS	72
4.6	Shutting down or rebooting the robot controller	73
4.7	Switching off the power to save energy	76
4.8	Switching drives on/off	77
4.9	Switching the robot controller off	77
4.10	Setting the user interface language	78
4.11	Creating a screenshot on the smartPAD	78
4.12	Online documentation and help for messages	78
4.12.1	Calling online documentation	78
4.12.2	Calling help for the messages	79
4.13	Changing user group	82
4.14	User groups	83
4.15	Changing operating mode	84
4.16	Coordinate systems	85
4.17	Jogging the robot	86
4.17.1	Setting the jogging options	87
4.17.1.1	“General” tab	87
4.17.1.2	“Keys” tab	88
4.17.1.3	Mouse tab	89
4.17.1.4	“KCP pos.” tab	90
4.17.1.5	Cur. tool/base tab	91
4.17.1.6	“Collision detection” tab	91

4.17.2	Setting the jog override	92
4.17.3	Selecting the tool and base	93
4.17.4	Axis-specific jogging with the jog keys	93
4.17.5	Cartesian jogging with the jog keys	94
4.17.6	Configuring the Space Mouse	94
4.17.7	Defining the alignment of the Space Mouse	96
4.17.8	Cartesian jogging with the Space Mouse	97
4.17.9	Temporarily deactivating the Space Mouse	97
4.17.10	Incremental jogging	98
4.17.11	Aligning the tool with a base	99
4.17.12	Backward motion using the jog keys	101
4.17.12.1	Backward motion using the jog keys – Overview	101
4.17.12.2	Recording in buffer	102
4.17.12.3	Executing motions backwards (using jog keys)	104
4.18	Jogging external axes	104
4.19	Bypassing workspace monitoring	105
4.20	Display functions	106
4.20.1	Displaying the actual position	106
4.20.1.1	Window Actual position, view Cartesian	106
4.20.1.2	Window Actual position, view Axis-specific	107
4.20.2	Displaying digital inputs/outputs	109
4.20.3	Displaying analog inputs/outputs	111
4.20.4	Displaying inputs/outputs for Automatic External	111
4.20.5	Displaying and modifying the value of a variable	112
4.20.6	Displaying the state of a variable	114
4.20.7	Displaying the variable overview and modifying variables	114
4.20.8	Displaying cyclical flags	116
4.20.9	Displaying flags	117
4.20.10	Displaying counters	118
4.20.11	Displaying timers	119
4.20.12	Displaying information about the robot and robot controller	120
4.20.13	Displaying/editing robot data	121
4.20.14	Measuring and displaying energy consumption	123
4.20.14.1	$ENERGY_MEASURING: starting and stopping consumption measurement	125
4.20.14.2	$ENERGYMODULE: energy model available for robot?	126
4.20.14.3	$ENERGY_CONFIG_STATE: Status of the energy model for each axis	127
4.20.14.4	$ENERGY_PERIOD: energy consumption of the last 60 min	128
4.20.14.5	$ENERGY_TOTAL: energy consumption since last cold start	129
4.20.14.6	$ENERGY_INTERIM: energy consumption between time stamps	129
4.20.14.7	INTERIMENERGY: setting a time stamp for the measurement of energy consumption	130
4.21	Automatic firmware update for hardware components	131
4.21.1	Starting the update immediately	132
4.21.2	Starting the update later (manually)	133
4.21.3	“Firmware / Hardware Manager” window	133
4.22	Displaying the battery state	134
5	**Start-up and recommissioning**	**137**
5.1	Safety acceptance overview (without safety option)	137

5.2 Switching on the robot controller with the kinematic system for the first time... 137
5.3 Protecting the Windows system... 141
5.4 Changing the initial password for “KukaUser”... 142
5.5 Changing the initial password for “KukaSystem” and “target”... 143
5.5.1 Changing the password: return values... 144
5.6 Restricting remote access to the robot controller... 146
5.7 Start-up wizard... 148
5.8 Modifying the machine data configuration... 148
5.9 Defining hardware options... 149
5.10 Preventing an unexpected start of the peripheral devices... 151
5.10.1 Behavior with/without additional enabling... 151
5.10.2 Configuring additional enabling... 152
5.10.3 $CRIT_PERI_ACK... 153
5.10.4 $CRIT_PERI_ACK_REQ... 154
5.11 Changing the safety ID of the PROFINET device... 156
5.12 Switching safe Cartesian monitoring on/off... 157
5.13 Jogging the robot without a higher-level safety controller... 158
5.14 Checking the activation of the positionally accurate robot model... 159
5.15 Activating palletizing mode... 160
5.16 Mastering... 161
5.16.1 Mastering methods... 162
5.16.2 Moving axes to the pre-mastering position using mastering marks... 164
5.16.3 Moving axes to the pre-mastering position using the probe... 166
5.16.4 Mastering LEDs... 167
5.16.5 Mastering with the SEMD... 168
5.16.5.1 First mastering (with SEMD)... 169
5.16.5.2 Teach offset (with SEMD)... 172
5.16.5.3 Checking load mastering with offset (with SEMD)... 173
5.16.6 Mastering with the dial gauge... 175
5.16.7 Mastering external axes... 176
5.16.8 Reference mastering... 177
5.16.9 Mastering with the MEMD and mark... 178
5.16.9.1 Moving A6 to the mastering position (with line mark)... 179
5.16.9.2 First mastering (with MEMD)... 180
5.16.9.3 Teach offset (with MEMD)... 183
5.16.9.4 Checking load mastering with offset (with MEMD)... 184
5.16.10 Manually unmastering axes... 186
5.17 Modifying software limit switches... 186
5.18 Calibration... 189
5.18.1 Defining the tool direction... 189
5.18.2 Introduction to TOOL calibration... 190
5.18.3 Calibrate TOOL or enter it numerically (tool/workpiece on flange)... 191
5.18.4 Introduction to BASE calibration... 192
5.18.5 Calibrate BASE or enter it numerically (base/fixed tool)... 192
5.18.6 Introduction to calibration of external kinematic systems... 193
5.18.7 Calibrating or numerically entering the root point of an external kinematic system... 194
5.18.8 “Tool/base management” window... 195
5.18.8.1 Tool/base management window – “Overview” area... 195

5.18.8.2	Icons in the “Overview” area	196
5.18.8.3	Tool/base management window – “Editing” area	197
5.18.8.4	Tool/base management window – “Calibration” area	198
5.18.9	Overview of calibration methods	200
5.18.9.1	XYZ 4-point method	200
5.18.9.2	XYZ 3-point method	202
5.18.9.3	XYZ Reference method	202
5.18.9.4	XYZ method	203
5.18.9.5	ABC world method	204
5.18.9.6	ABC 2-point method	205
5.18.9.7	3-point method	206
5.18.9.8	Indirect method	208
5.18.9.9	Method Root point	209
5.18.10	Linear unit	210
5.18.10.1	Checking whether the linear unit needs to be calibrated	211
5.18.10.2	Calibrating the linear unit	212
5.18.10.3	Entering the linear unit numerically	213
5.19	Load data	213
5.19.1	Load data parameters	214
5.19.2	Entering payload data numerically	215
5.19.3	Entering supplementary load data numerically	216
5.19.4	Importing supplementary load data for A3	217
5.19.5	Load data verification	219
5.20	Exporting/importing long texts	221
5.21	Adapting the MAMES values after exchanging the in-line wrist	223
5.22	Maintenance handbook	224
5.22.1	Logging maintenance	225
5.22.2	Displaying a maintenance log	226
6	**Configuration**	**229**
6.1	Configuring the KUKA Line Interface (KLI)	229
6.1.1	Configuring the Windows interface (without field bus)	229
6.1.2	Configuring the field bus interface and creating the Windows interface	231
6.1.3	Displaying ports of the Windows interface or enabling an additional port	234
6.1.4	Displaying or modifying filters	235
6.1.5	Error display in the Address and Subnet boxes	236
6.2	Displaying the subnet configuration of the robot controller	236
6.3	Reconfiguring the I/O driver	237
6.4	Configuring safe axis monitoring functions	238
6.4.1	Parameter: Braking time	240
6.4.2	Checking the limits for the maximum axis velocity in T1 mode	242
6.5	Checking the values for the safe axis monitoring functions	244
6.6	Checking the safety configuration of the robot controller	245
6.7	Activation code of the safety configuration	246
6.8	Exporting the safety configuration (XML export)	246
6.8.1	Changing the directory for the export	247
6.8.2	Switching off automatic export	248
6.9	Protecting a digital output against manual modifications	248
6.10	Configuring the variable overview	249

6.11 Changing the password ... 251
6.12 Displaying and modifying user rights ... 251
6.12.1 Information about function groups in this documentation ... 252
6.12.2 Overview of function groups ... 253
6.13 Enabling or disabling operating modes for specific user groups ... 255
6.14 Energy saving mode ($ECO_LEVEL) ... 256
6.15 Configuring workspaces ... 257
6.15.1 Configuring Cartesian workspaces ... 258
6.15.2 Modes for Cartesian workspaces ... 262
6.15.3 Configuring axis-specific workspaces ... 263
6.15.4 Modes for axis-specific workspaces ... 266
6.16 Workspace for the KR DELTA ... 266
6.17 Defining limits for reteaching ... 268
6.18 Warm-up ... 269
6.18.1 Warm-up sequence ... 269
6.18.2 Configuring warm-up ... 270
6.19 Defining calibration tolerances ... 272
6.20 Configuring “user list” with variables for inline forms ... 273
6.20.1 Example of a “user list” ... 273
6.21 Configuring backward motion ... 275
6.22 Configuring Automatic External ... 276
6.22.1 Configuring CELL.SRC ... 277
6.22.2 Configuring Automatic External inputs/outputs ... 278
6.22.3 Automatic External: inputs ... 279
6.22.3.1 PGNO_TYPE ... 279
6.22.3.2 REFLECT_PROG_NR ... 280
6.22.3.3 PGNO_LENGTH ... 281
6.22.3.4 PGNO_FBIT ... 281
6.22.3.5 PGNO_PARITY ... 282
6.22.3.6 PGNO_VALID ... 282
6.22.3.7 $EXT_START ... 283
6.22.3.8 $MOVE_ENABLE ... 283
6.22.3.9 $CHCK_MOVENA ... 284
6.22.3.10 $CONF_MESS ... 284
6.22.3.11 $DRIVES_OFF ... 285
6.22.3.12 $DRIVES_ON ... 285
6.22.3.13 $I_O_ACT ... 285
6.22.4 Odd / even parity ... 286
6.22.5 Automatic External: outputs ... 287
6.22.5.1 $RC_RDY1 ... 287
6.22.5.2 $ALARM_STOP ... 287
6.22.5.3 $USER_SAF ... 288
6.22.5.4 $PERI_RDY ... 288
6.22.5.5 $ROB_CAL ... 288
6.22.5.6 $I_O_ACTCONF ... 289
6.22.5.7 $STOPMESS ... 289
6.22.5.8 PGNO_FBIT_REFL ... 289
6.22.5.9 $ALARM_STOP_INTERN ... 290
6.22.5.10 $PRO_ACT ... 290

6.22.5.11 PGNO_REQ ... 291
6.22.5.12 APPL_RUN ... 291
6.22.5.13 $PRO_MOVE ... 292
6.22.5.14 $IN_HOME ... 292
6.22.5.15 $IN_HOME1 … $IN_HOME5 ... 293
6.22.5.16 $ON_PATH ... 293
6.22.5.17 $NEAR_POSRET ... 293
6.22.5.18 $ROB_STOPPED ... 294
6.22.6 Transmitting error numbers to the higher-level controller ... 294
6.22.7 Signal diagrams ... 296
6.23 Torque mode ... 302
6.23.1 Overview of torque mode ... 302
6.23.1.1 Using torque mode ... 302
6.23.1.2 Torque mode program example: setting A1 to “soft” in both directions ... 304
6.23.2 SET_TORQUE_LIMITS: activating torque mode ... 305
6.23.3 RESET_TORQUE_LIMITS: deactivating torque mode ... 308
6.23.4 Interpreter specifics ... 309
6.23.5 Diagnostic variables for torque mode ... 310
6.23.5.1 $TORQUE_AXIS_ACT ... 310
6.23.5.2 $TORQUE_AXIS_MAX_0 ... 311
6.23.5.3 $TORQUE_AXIS_MAX ... 311
6.23.5.4 $TORQUE_AXIS_LIMITS ... 311
6.23.5.5 $HOLDING_TORQUE ... 312
6.23.5.6 Comparison: $TORQUE_AXIS_ACT and $HOLDING_TORQUE ... 313
6.23.6 Other examples ... 313
6.23.6.1 Torque mode program example: setting axis to “soft” in both directions ... 313
6.23.6.2 Torque mode program example: avoiding damage in the event of collisions ... 314
6.23.6.3 Program example: torque mode in the interrupt ... 315
6.23.6.4 Torque mode program example: servo gun builds up pressure (robot program) ... 316
6.23.6.5 Torque mode program example: servo gun builds up pressure (submit program) ... 318
6.24 Event planner: configuring a data comparison ... 319
6.25 Brake test ... 319
6.25.1 Overview of the brake test ... 319
6.25.2 Sequence when testing a brake ... 321
6.25.3 Programs for the brake test ... 321
6.25.4 Overview of the brake test setup ... 322
6.25.4.1 Activating the brake test, defining the cycle time and axes ... 322
6.25.4.2 “Brake test configuration” window ... 323
6.25.4.3 Configuring input and output signals for the brake test ... 324
6.25.4.4 Signal diagram of the brake test – examples ... 327
6.25.4.5 Teaching positions for the brake test ... 328
6.25.4.6 Testing the sequence in the case of defective brakes ... 329
6.25.5 Performing a brake test ... 330
6.25.5.1 Performing a brake test for requested axes (cyclically via program) ... 330
6.25.5.2 Performing a brake test for active axes (manually) ... 332
6.25.5.3 Performing a brake test for further axes (e.g. couplable axes) ... 334
6.25.6 Automatic brake check ... 335

6.25.7 System functions for the brake test ... 336
6.25.7.1 GET_AXESMASK: requesting information about the axes for the brake test ... 336
6.25.7.2 GET_BRAKETEST_TIME: requesting time values for brake test ... 337

7 Program and project management ... 339

7.1 Creating a new program ... 339
7.2 Creating a new folder ... 339
7.3 Renaming a file or folder ... 339
7.4 Navigator file manager ... 340
7.4.1 Selecting filters ... 341
7.4.2 Displaying or modifying properties of files and folders ... 342
7.5 Selecting or opening a program ... 345
7.5.1 Selecting and deselecting a program ... 345
7.5.2 Opening a program ... 346
7.5.3 Toggling between the Navigator and the program ... 347
7.6 Structure of a KRL program ... 348
7.6.1 HOME position ... 348
7.7 Displaying/hiding program sections ... 349
7.7.1 Displaying/hiding the DEF line ... 349
7.7.2 Activating detail view ... 349
7.7.3 Activating/deactivating the line break function ... 350
7.7.4 Displaying folds ... 350
7.8 Editing programs ... 352
7.8.1 Deleting program lines ... 352
7.8.2 Selecting a line or range ... 352
7.8.3 Creating folds ... 352
7.8.4 Inserting a comment or stamp in a program ... 353
7.8.5 Commenting out program code ... 354
7.8.6 Display of comments in the program ... 355
7.8.7 Additional editing functions ... 356
7.9 Archiving and restoring data ... 357
7.9.1 Archiving overview ... 357
7.9.2 Archiving to a USB stick ... 358
7.9.3 Archiving on the network ... 359
7.9.4 Archiving the logbook ... 360
7.9.5 Restoring data ... 360
7.10 Project management ... 361
7.10.1 Pinning a project on the robot controller ... 361
7.10.2 Activating an existing project ... 361
7.10.3 Project management window ... 363
7.10.3.1 Projects tab ... 363
7.10.3.2 Restoration points tab ... 365
7.11 Backup Manager ... 366
7.11.1 Overview of Backup Manager ... 366
7.11.2 Backing up projects, option packages and RDC data manually ... 367
7.11.3 Manually restoring projects and option packages ... 368
7.11.4 Restoring RDC data manually ... 370
7.11.5 Configuring Backup Manager ... 371
7.11.5.1 “Backup configuration” tab ... 372

7.11.5.2 “Signal interface” tab ... 374

8 Program execution ... 377

8.1 Selecting the program run mode ... 377
8.1.1 Program run modes ... 377
8.2 Advance run ... 378
8.3 Block pointer ... 379
8.4 Setting the program override ... 381
8.5 Robot interpreter status indicator ... 382
8.6 Starting a program forwards (manual) ... 382
8.7 Starting a program forwards (automatic) ... 383
8.8 BCO run ... 383
8.9 Carrying out a block selection ... 384
8.10 Resetting a program ... 385
8.11 Starting Automatic External mode ... 385
8.12 Backward motion using the Start backwards key ... 386
8.12.1 Executing motions backwards (using the “Start backwards” key) ... 386
8.12.2 Functional principle and characteristics of backward motion ... 387
8.12.2.1 Response in the case of subprograms ... 388
8.12.2.2 Approximate positioning response ... 388
8.12.2.3 Response in the case of weave motions ... 390
8.12.2.4 Switching from backwards to forwards ... 390
8.12.3 System variables with changed meaning ... 391
8.12.4 Comparison of “Start backwards”/backwards using the jog keys ... 391
8.13 Making a subprogram invisible ... 392
8.14 Collision detection ... 392
8.14.1 Overview of collision detection ... 392
8.14.1.1 Resuming motion after a collision ... 393
8.14.1.2 Determining peak values for collision detection ... 394
8.14.2 Configuring collision detection for jog mode ... 394
8.14.2.1 “Collision detection - Jogging configuration” window ... 395
8.14.3 Configuring collision detection for program mode – overview ... 396
8.14.4 Filling data sets via the configuration window ... 396
8.14.4.1 “Collision detection - Data set overview” window ... 397
8.14.4.2 “Collision detection - Data set configuration” window ... 398
8.14.4.3 Defining offsets / Offset settings window ... 399
8.14.5 Filling data sets via a program in teach mode ... 400
8.14.5.1 “Collision detection - Data set teach view” window ... 401
8.14.6 Activating general collision detection ... 402
8.14.7 Activating collision detection for a motion ... 403
8.14.8 Displaying current values / Collision detection - Display window ... 405
8.14.9 Inline forms only for compatibility with previous versions ... 406
8.14.9.1 Inline form SaveMax ... 406
8.14.9.2 Inline forms SetDefault, SetLimits ... 407
8.14.9.3 Saving default values for collision detection in $custom.dat ... 407
8.14.10 Program example: search run via collision detection ... 408
8.14.10.1 Main program ... 408
8.14.10.2 Search program ... 409
8.14.10.3 Return motion program ... 411

8.14.11 $COLLMON_STARTUP_MAX: adapting the inactive collision detection phase after start of motion ... 412

9 Basic principles of motion programming ... 415

9.1 Overview of motion types ... 415
9.2 Motion type PTP ... 415
9.3 Motion type LIN ... 416
9.4 Motion type CIRC ... 416
9.5 Approximate positioning ... 417
9.6 Orientation control LIN, CIRC ... 418
9.6.1 Combinations of $ORI_TYPE and $CIRC_TYPE ... 419
9.7 Spline motion type ... 421
9.7.1 Velocity profile for spline motions ... 423
9.7.2 BCO run with spline motions via the Block selection button ... 424
9.7.3 BCO run with spline motions after program modification ... 426
9.7.3.1 BCO run following modification to the current spline block ... 426
9.7.3.2 BCO run following modification of another spline block ... 427
9.7.4 Modifications to spline blocks ... 429
9.7.5 Approximation of spline motions ... 432
9.7.6 Replacing an approximated CP motion with a spline block ... 432
9.7.6.1 SLIN-SPL-SLIN transition ... 435
9.8 Orientation control for CP spline motions ... 436
9.8.1 SCIRC: reference system for the orientation control ... 438
9.8.2 SCIRC: orientation behavior ... 439
9.8.2.1 SCIRC: orientation behavior – example: auxiliary point ... 441
9.8.2.2 SCIRC: orientation behavior – example: end point ... 442
9.9 Circular angle ... 443
9.10 Status and Turn ... 444
9.10.1 Status ... 445
9.10.2 Turn ... 447
9.11 Singularities ... 448

10 Programming with inline forms ... 451

10.1 Instructions for programming ... 451
10.2 Names in inline forms ... 451
10.3 Programming PTP, LIN and CIRC motions ... 452
10.3.1 Inline form PTP: programming a PTP motion ... 452
10.3.2 Inline form LIN: programming a LIN motion ... 453
10.3.3 Inline form CIRC: programming a CIRC motion ... 454
10.3.4 Option window “Frames” ... 455
10.3.5 Option window “Motion parameters” (LIN, CIRC, PTP) ... 456
10.3.6 Near range approximate positioning ... 457
10.3.6.1 Activating near range approximate positioning for inline forms ... 458
10.3.6.2 Programming near range approximate positioning via an inline form ... 459
10.4 Inline form HOP: programming a combined motion ... 460
10.4.1 Indirect transfer motion ... 463
10.4.2 Option window for the HOP motion parameters ... 463
10.4.3 Option window for HOP Trigger ... 465
10.5 Programming spline motions ... 466
10.5.1 Programming tips for spline motions ... 466

10.5.2 Programming a spline block ... 468
10.5.2.1 Inline form “Spline”: programming a CP spline block ... 468
10.5.2.2 Inline form “PTP spline”: programming a PTP spline block ... 469
10.5.2.3 Option window “Motion parameters” (CP spline block) ... 471
10.5.2.4 Option window “Motion parameters” (PTP spline block) ... 472
10.5.3 Programming segments for a spline block ... 472
10.5.3.1 Inline form “SPL”: Programming the SPL segment ... 472
10.5.3.2 Inline form “SLIN”: Programming the SLIN segment ... 474
10.5.3.3 Inline form “SCIRC”: Programming the SCIRC segment ... 475
10.5.3.4 Inline form “SPTP”: Programming the SPTP segment ... 477
10.5.3.5 Option window “Motion parameters” (CP spline segment) ... 479
10.5.3.6 Option window “Motion parameters” (SPTP) ... 480
10.5.4 Programming individual spline motions ... 481
10.5.4.1 Inline form “SLIN”: Programming an individual SLIN motion ... 481
10.5.4.2 Option window “Motion parameters” (SLIN) ... 483
10.5.4.3 Inline form “SCIRC”: Programming an individual SCIRC motion ... 483
10.5.4.4 Option window “Motion parameters” (SCIRC) ... 485
10.5.4.5 Inline form “SPTP”: Programming an individual SPTP motion ... 486
10.5.5 Programming a trigger for spline ... 488
10.5.5.1 Programming a trigger at the spline segment or individual spline instruction ... 488
10.5.5.2 Inline form TRIGGER WHEN PATH: programming a trigger in the spline block ... 490
10.5.6 Programming a conditional stop for spline ... 493
10.5.6.1 Programming a conditional stop at the spline segment or individual spline instruction ... 493
10.5.6.2 Inline form STOP WHEN PATH: programming a conditional stop for a an individual spline instruction or spline block ... 495
10.5.6.3 Stop condition: example and braking characteristics ... 496
10.5.7 Programming a constant velocity range for a CP spline block ... 498
10.5.7.1 Block selection to a constant velocity range ... 500
10.5.7.2 Maximum limits of a constant velocity range ... 500
10.5.8 Teaching “Path” for trigger, conditional stop or constant velocity range ... 501
10.6 Displaying the distance between points ... 502
10.7 Modifying programmed motions ... 502
10.7.1 Modifying motion parameters ... 502
10.7.2 Modifying blocks of motion parameters ... 503
10.7.3 Re-teaching a point ... 503
10.7.4 Transforming blocks of coordinates ... 503
10.7.4.1 “Axis mirroring” window ... 507
10.7.4.2 “Transform - Axis Specific” window ... 508
10.7.4.3 “Transform - Cartesian Base” window ... 509
10.8 Global points ... 510
10.8.1 Global points – introduction ... 510
10.8.2 Creating a global point (creating new point or converting local point) ... 511
10.8.3 Modifying global points individually ... 512
10.8.4 Displaying and modifying global points in the overview ... 513
10.8.5 Adding files in the Global points overview ... 515
10.9 Programming logic instructions ... 516
10.9.1 Inline form WAIT: Programming a wait time ... 516

10.9.2 WAIT FOR inline form: programming a signal-dependent wait function 517
10.9.3 Inline form OUT: Setting a digital output 518
10.9.4 Inline form PULSE: Setting a pulse output 519
10.9.5 Switching on the path - SYN OUT 520
10.9.5.1 Inline form SYN OUT, option START/END 520
10.9.5.2 Inline form SYN OUT, option PATH 523
10.9.6 Inline form SYN PULSE: Setting a pulse on the path 525
10.9.7 Setting an analog output 526
10.9.7.1 Inline form “ANOUT” (static) 527
10.9.7.2 Inline form “ANOUT” (dynamic) 527
10.9.8 Inline form TIMER: Programming a timer 528
10.9.8.1 $TIMER_FLAG 529
10.9.9 Inline form “Variable declarations” 530
10.9.10 Inline form “Interrupt declaration” 531
10.9.11 Inline form “Switch the interrupt” 533
10.9.12 Inline form IF ... THEN: Programming a conditional branch 534
10.9.13 Inline form WHILE ... ENDWHILE: Programming a rejecting loop 535
10.9.14 Inline form REPEAT ... UNTIL: Programming a post-test loop 537
10.9.15 Inline form LOOP ... ENDLOOP: Programming an endless loop 538
10.9.16 Inline form FOR ... ENDFOR: Programming a counting loop 540
10.9.17 Inline form SWITCH ... CASE: Programming a multiple branch 541
10.9.18 Modifying a logic instruction 543

11 KRL programming 545

11.1 Instructions for programming 545
11.2 Symbols and fonts 545
11.3 Important KRL terms 546
11.3.1 SRC files and DAT files 546
11.3.2 Naming conventions and keywords 546
11.3.3 Data types 547
11.3.4 Areas of validity 549
11.3.4.1 Making subprograms, functions and interrupts available globally 549
11.3.4.2 Making variables, constants, signals and user data types available globally 550
11.3.5 CONST: constants 551
11.4 Variables and declarations 551
11.4.1 DECL: declaring variables, arrays and constants 551
11.4.2 ENUM: defining an enumeration type 553
11.4.3 STRUC: defining a structure type 554
11.5 Motion programming: PTP, LIN, CIRC 555
11.5.1 PTP: motion programming 555
11.5.2 LIN: motion programming 556
11.5.3 CIRC: motion programming 557
11.5.4 PTP_REL: relative motion programming 558
11.5.5 LIN_REL: relative motion programming 559
11.5.6 CIRC_REL: relative motion programming 561
11.5.7 Approximation parameters for PTP, LIN, CIRC and …_REL 562
11.5.8 REL motions for infinitely rotating axes 564
11.6 Motion programming: spline 565
11.6.1 SPLINE ... ENDSPLINE: programming a CP spline block 565

11.6.2 PTP_SPLINE ... ENDSPLINE: programming a PTP spline block ... 566
11.6.3 SLIN: motion programming ... 567
11.6.4 SCIRC: motion programming ... 569
11.6.5 SPL: motion programming ... 570
11.6.6 SPTP: motion programming ... 570
11.6.7 SLIN_REL: relative motion programming ... 572
11.6.8 SCIRC_REL: relative motion programming ... 573
11.6.9 SPL_REL: relative motion programming ... 575
11.6.10 SPTP_REL: relative motion programming ... 576
11.6.11 System variables to be used in the WITH line of spline commands ... 577
11.6.12 TIME_BLOCK: programming a time block for spline ... 578
11.6.13 CONST_VEL: programming a constant velocity range for a CP spline block ... 582
11.6.13.1 Block selection to a constant velocity range ... 583
11.6.13.2 Maximum limits of a constant velocity range ... 584
11.6.13.3 System variables for CONST_VEL ... 584
11.6.14 STOP WHEN PATH: programming a conditional stop for spline ... 586
11.6.14.1 Stop condition: example and braking characteristics ... 587
11.6.15 $EX_AX_IGNORE ... 589
11.6.16 $STRIKE_OUT_DEFAULT: preventing velocity drops in the CP spline block ... 590
11.7 $VEL_APPL: changing the velocity of a CP motion in the main run ... 591
11.8 Program execution control ... 594
11.8.1 CONTINUE: preventing an advance run stop ... 594
11.8.2 EXIT: exit a loop ... 594
11.8.3 FOR ... TO ... ENDFOR: programming a counting loop ... 595
11.8.4 GOTO: jump to position in the program ... 596
11.8.5 HALT: stopping a program ... 597
11.8.6 IF ... THEN ... ENDIF: programming a conditional branch ... 597
11.8.7 LOOP ... ENDLOOP: programming an endless loop ... 598
11.8.8 ON_ERROR_PROCEED: skipping an instruction that causes an interpreter error ... 599
11.8.8.1 Handlable/non-handlable errors ... 600
11.8.8.2 ON_ERROR_PROCEED with motion instructions ... 600
11.8.8.3 $ERR ... 601
11.8.8.4 ERR_RAISE: subsequently generate suppressed message ... 604
11.8.8.5 Program examples: ON_ERROR_PROCEED with ERR_RAISE ... 605
11.8.9 REPEAT ... UNTIL: programming a post-test loop ... 607
11.8.10 SWITCH ... CASE ... ENDSWITCH: programming a multiple branch ... 607
11.8.11 WAIT FOR … : waiting until a condition has been met ... 609
11.8.12 WAIT SEC … : programming a wait time ... 609
11.8.13 WHILE ... ENDWHILE: programming a rejecting loop ... 610
11.9 Inputs/outputs ... 611
11.9.1 ANIN ON / ANIN OFF: cyclically reading an analog input ... 611
11.9.2 ANOUT ON / ANOUT OFF: writing cyclically to an analog output ... 612
11.9.3 PULSE: setting a pulse output ... 613
11.9.4 SIGNAL: signal declaration for inputs/outputs ... 617
11.10 Subprograms and functions ... 618
11.10.1 Calling a subprogram ... 618
11.10.2 Calling a function ... 619
11.10.3 DEFFCT ... ENDFCT: defining a function ... 619

11.10.4 RETURN: jump back to the calling program ... 620
11.10.5 Transferring parameters to a subprogram or function ... 621
11.10.6 Transferring a parameter to a different data type ... 625
11.11 Interrupt programming ... 625
11.11.1 INTERRUPT … DECL … WHEN … DO … : Declare interrupt ... 625
11.11.2 INTERRUPT ON/OFF: activating or deactivating an interrupt ... 628
11.11.3 INTERRUPT DISABLE/ENABLE: disabling or enabling an interrupt ... 629
11.11.4 BRAKE: stopping the robot from an interrupt program ... 631
11.11.5 RESUME: cancelling interrupt programs ... 632
11.12 Path-related switching actions (=Trigger) ... 634
11.12.1 TRIGGER WHEN DISTANCE ... 634
11.12.2 TRIGGER WHEN PATH ... 638
11.12.2.1 Reference point for approximate positioning – overview ... 642
11.12.2.2 Reference point for homogenous approximate positioning ... 642
11.12.2.3 Reference point for mixed approximate positioning (spline) ... 644
11.12.2.4 Reference point for mixed approximate positioning (LIN/CIRC/PTP) ... 645
11.12.3 Limits for functions in the KRL trigger ... 645
11.12.4 System variables for working with PATH triggers ... 646
11.12.4.1 $DIST_NEXT ... 646
11.12.4.2 $DIST_LAST ... 647
11.12.4.3 Response when direction changes ... 647
11.13 Communication ... 648
11.14 Operators ... 649
11.14.1 Arithmetic operators ... 649
11.14.2 Geometric operator ... 650
11.14.2.1 Sequence of the operands ... 651
11.14.2.2 Example of a double operation ... 652
11.14.3 Relational operators ... 653
11.14.4 Logic operators ... 654
11.14.5 Bit operators ... 655
11.14.6 Priority of the operators ... 657
11.15 Mathematical standard functions ... 658
11.16 System functions ... 660
11.16.1 DELETE_BACKWARD_BUFFER: preventing backward start ... 660
11.16.2 FORWARD: converting axis angles to a Cartesian position ... 661
11.16.3 GET_SIG_INF: reading information about signal declaration ... 662
11.16.4 INV_POS: calculating an inverse position ... 664
11.16.5 INVERSE: converting a Cartesian position to axis angles ... 665
11.16.6 ROB_STOP: prevent robot motions ... 667
11.16.7 ROB_STOP_RELEASE: permit robot motions again ... 669
11.16.8 SET_BRAKE_DELAY: reducing the delay time for applying the brakes ... 669
11.16.9 TIMER_LIMIT: timer that only runs with an active interpreter ... 672
11.16.10 VARSTATE: polling the state of a variable ... 672
11.17 Editing string variables ... 674
11.17.1 StrTo… : converting a character string to a different data type ... 674
11.17.1.1 StrToAXIS: converting character string to AXIS ... 674
11.17.1.2 StrToE3AXIS: converting character string to E3AXIS ... 675
11.17.1.3 StrToE6AXIS: converting character string to E6AXIS ... 676
11.17.1.4 StrToFRAME: converting character string to FRAME ... 677

11.17.1.5 StrToPOS: converting character string to POS ... 678
11.17.1.6 StrToE3POS: converting character string to E3POS ... 679
11.17.1.7 StrToE6POS: converting character string to E6POS ... 680
11.17.1.8 StrToBOOL: converting character string to BOOL ... 681
11.17.1.9 StrToINT: converting character string to INT ... 682
11.17.1.10 StrToREAL: converting character string to REAL ... 683
11.17.1.11 StrToSTRING: converting nested character string to simple character string (CHAR) ... 683
11.17.2 StrDeclLen: string variable length in the declaration ... 685
11.17.3 StrLen: string variable length after initialization ... 685
11.17.4 StrClear: deleting the contents of a string variable ... 686
11.17.5 StrAdd: extending a string variable ... 686
11.17.6 StrFind: searching a string variable ... 687
11.17.7 StrComp: comparing the contents of string variables ... 688
11.17.8 StrCopy: copying a string variable ... 689

12 Submit interpreter ... 691

12.1 Function of the submit interpreters ... 691
12.2 Status indicator for the submit interpreter ... 692
12.3 Operation of submit interpreter ... 693
12.3.1 Automatic starting of submit interpreters (on cold start) ... 693
12.3.2 Manual control of submit interpreters ... 694
12.3.2.1 Stopping all submit interpreters ... 694
12.3.2.2 Deselecting all submit interpreters ... 694
12.3.2.3 Starting all submit interpreters ... 695
12.3.2.4 Stopping, resetting or deselecting individual submit interpreters ... 695
12.3.2.5 Starting individual submit interpreters ... 696
12.3.2.6 Starting the system submit interpreter individually ... 696
12.3.3 “Submit interpreter” window ... 697
12.3.3.1 “Current display/assignment” tab ... 697
12.3.3.2 “Cold start configuration” tab ... 698
12.4 Submit interpreter programming ... 699
12.4.1 Creating a new SUB program ... 699
12.4.2 Editing the sps.sub program ... 700
12.4.3 KRL instructions not allowed or subject to restrictions ... 702
12.4.4 Subprograms ... 702
12.4.5 Inputs/outputs and communication ... 703
12.4.6 System variables ... 703
12.4.6.1 Changes in relation to Single Submit mode ... 704
12.4.6.2 $INTERPRETER ... 705
12.4.6.3 $PROG_INFO ... 705
12.4.7 CWRITE: changes in relation to Single Submit mode ... 707
12.5 Overview: changes of state due to actions in the menu ... 709

13 Diagnosis ... 713

13.1 Logbook ... 713
13.1.1 Displaying the logbook ... 713
13.1.2 “Log” tab ... 713
13.1.3 Filter tab ... 715
13.1.4 Configuring the logbook ... 715

13.2 Display the moment load of the robot ... 716
13.3 Displaying the caller stack ... 717
13.4 Displaying interrupts ... 718
13.5 Displaying diagnostic data about the kernel system ... 720
13.6 Automatically compressing data for error analysis (diagnosis package “KRCDiag”) ... 720

14 Installation ... 723

14.1 System requirements ... 723
14.2 Installing additional software ... 723
14.3 KSS update ... 725
14.3.1 Update from USB stick ... 725
14.3.2 Update from the network ... 726

15 Appendix ... 729

15.1 Checklists for safety acceptance ... 729
15.1.1 Precondition for safety acceptance based on the checklists ... 729
15.1.2 Checklist: Robot and system ... 729
15.1.3 Checklist: Safety functions ... 730
15.1.4 Checklist: Maximum axis velocities for T1 ... 732
15.2 Assignment of functions and function groups ... 734
15.2.1 Menu item File ... 735
15.2.2 Menu item Configuration ... 736
15.2.2.1 Automatic External ... 737
15.2.2.2 I/O drivers ... 737
15.2.2.3 Cartesian workspaces/Axis-specific workspaces ... 737
15.2.2.4 Event planner ... 737
15.2.2.5 Point coordinate correction limit ... 738
15.2.3 Menu item Display ... 738
15.2.3.1 Variable display ... 738
15.2.3.2 Variable overview ... 739
15.2.4 Menu item Diagnosis ... 739
15.2.4.1 Trace ... 739
15.2.4.2 TraceLogbook ... 739
15.2.5 Menu item Start-up ... 740
15.2.5.1 Robot data ... 741
15.2.5.2 Network configuration ... 741
15.2.5.3 Additional software ... 741
15.2.6 Menu item Shutdown ... 742
15.2.7 Menu item Help ... 742
15.2.8 Navigator ... 742
15.2.9 Navigator: Edit menu ... 743
15.2.10 Editor: button bar ... 744
15.2.11 “Project management” window ... 744
15.2.12 Jog options window: Tabs ... 744
15.2.13 Wait messages ... 745
15.2.14 Editor: Edit menu ... 745
15.2.15 Editor: Commands menu ... 746
15.2.16 smartHMI: status bar ... 747
15.2.17 smartHMI: left sidebar ... 748

15.2.18 smartHMI: right sidebar 748

16 KUKA Service 751

16.1 Requesting support 751

16.2 KUKA Customer Support 751

Index 753

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 KUKA products, we recommend the training courses offered by 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 or smartPAD pro (if used)
- Documentation for the System Software
- Instructions for options and accessories
- Spare parts overview in KUKA Xpert

Each set 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
- On the KUKA smartHMI, in the main menu under **Help** > **Info**, tab **Licenses**



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

2 Product description

2.1 Product description

Description

The KUKA System Software (KSS) provides all the functions required for operator control of KUKA robot systems.

Functions

- Path planning
- I/O management
- Touchscreen user interface KUKA.smartHMI
- Simple programming with inline forms
- Expert programming with KRL (KUKA Robot Language)
- Data and file management
- Diagnostic functions
- User rights management
- Integrated safety PLC
- Safety-oriented functions and safety-oriented communication in discrete dual-channel technology

  Or direct safety-oriented communication to the safety PLC via EtherNet-based protocols
- Numerous other functions

2.2 Overview of the industrial robot

The industrial robot consists of the following components:

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

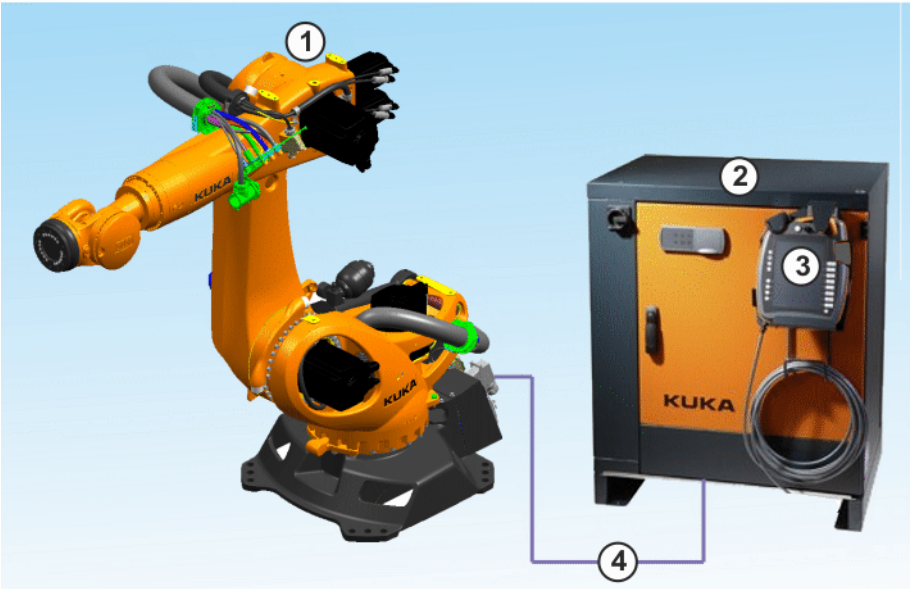


Fig. 2-1: Example of an industrial robot

1	Manipulator	3	KUKA smartPAD
2	Robot controller	4	Connecting cables

2.3 Network connections

Interfaces for “Performance” system board

The following diagram shows the Ethernet interfaces on the “Performance” system board.

- The interfaces shown in green are active in the standard scope of supply.
- Some interfaces must be activated in the software if they are to be used. This is carried out via separately available option packages. The option packages are listed in the table.

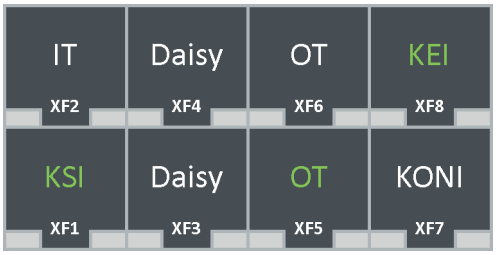


Fig. 2-2: Overview of interfaces for “Performance” system board

XF1	KUKA Service Interface	Standard scope of supply
XF2	KUKA Line Interface (IT)	KUKA.OptionalLineInterface IT
XF3	DaisyChain	Standard scope of supply
	KUKA Line Interface (OT)	KUKA.EthernetSwitch Quad-Port
XF4	DaisyChain	Standard scope of supply
	KUKA Line Interface (OT)	KUKA.EthernetSwitch Quad-Port
XF5	KUKA Line Interface (OT)	Standard scope of supply

XF6	KUKA Line Interface (OT)	KUKA.EthernetSwitch Dual-Port / KUKA.EthernetSwitch Quad-Port
XF7	KUKA Optional Network Interface	KUKA.OptionalNetworkInterface
XF8	KUKA Extension Interface (EtherCAT)	Standard scope of supply

2.4 System requirements

The System Software 8.7 can be run on the following robot controller:

- KR C5
- Windows 10 IoT, V5.1.x or higher
- With “Performance” system board

2.5 KUKA USB sticks

Special USB sticks from KUKA exist for the KR C5 robot controller. The sticks are available in the following variants:

- Bootable variant, exclusively in conjunction with the software KUKA.RecoveryUSB
  Color: Gray
- Non-bootable variant for data backup
  Color: Orange



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

NOTICE

Data loss due to USB sticks from third-party manufacturers
Data may be lost if USB sticks from manufacturers other than KUKA are used for activities on the robot controller.

- For activities on the robot controller requiring a USB stick, use a KUKA stick.
  The KUKA sticks are validated for use with the robot controller.

2.6 Intended use and misuse

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

3 Safety

3.1 General

3.1.1 Disclaimer

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 intended 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, especially those affecting safety, must be rectified immediately.

Safety information

Information about safety may not be construed against the manufacturer. 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 the manufacturer. Unauthorized modifications will result in the loss of warranty and liability claims.

Additional components (tools, software, etc.), not supplied by the manufacturer, 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 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.

3.1.3 Terms in the “Safety” chapter

STOP 0, STOP 1 and STOP 2 are the stop definitions according to EN 60204-1:2018.

Term	Description
Axis range	Range within which the axis 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.
AUT	Automatic mode Operating mode for program execution. The robot moves at the programmed velocity.
AUT EXT	Automatic mode, external Operating mode for program execution. The robot is moved at the programmed velocity by a higher-level controller.

User	The user of the industrial robot can be the management, employer or delegated person responsible for use of the industrial robot.
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.
Danger zone	The danger zone consists of the workspace and the stopping distances of the manipulator and external axes (optional).
KSS	KUKA System Software
KUKA smartPAD	see “smartPAD”
KUKA smartPAD-2	see “smartPAD”
Manipulator	The robot arm and the associated electrical installations
External EMERGENCY STOP device	Triggering an external EMERGENCY STOP affects the robot controllers in the system that are integrated into the external EMERGENCY STOP circuit. **Notice:** Triggering an external EMERGENCY STOP does not (!) set the output signal for the local EMERGENCY STOP.
EMERGENCY STOP device local	• EMERGENCY STOP device on the smartPAD • Optional: additional EMERGENCY STOP device on the front door of the cabinet • Optional: additional EMERGENCY STOP device on XG58 Triggering a local EMERGENCY STOP directly affects the robot controller to which this EMERGENCY STOP is connected.
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 execute a stop, but rather monitors whether the axes are at a standstill. If these are moved during the safe operational stop, a safety 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 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. • T1: The power supply to the brakes is switched off (SBC/Safe Brake Control signal) as soon as all axes are at a standstill or after 680 ms at the latest. 200 ms after SBC, the drives are switched off. • T2, AUT (KSS), AUT EXT (KSS), EXT (VSS): The drives are switched off after the configured braking time (default: 1.5 s). The power supply to the brakes is switched off 200 ms beforehand. The safety STOP 1 can also be triggered externally. **Note:** This stop is called “safety stop 1” in this document.
Safety STOP 1 – Drive Ramp Stop	Only relevant for safety options with BBRA (braking before restricted areas): 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 power supply to the brakes is switched off (SBC/Safe Brake Control signal) as soon as the robot stops or, at the latest, after the configured braking time (default: 1.5 s) plus an oscillation time of 500 ms. 200 ms after SBC, the drives are switched off. **Note:** This stop is called “Safety stop 1 DRS” in this document.
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. • T1: Safe standstill monitoring is activated as soon as all axes are at a standstill or after 680 ms at the latest. • T2, AUT (KSS), AUT EXT (KSS), EXT (VSS): After the configured braking time (default: 1.5 s), safe standstill monitoring is activated for all axes. 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	Teach pendant for the robot controller The smartPAD has all the operator control and display functions required for operating and programming the industrial robot. For robot controllers of the KR C5 series with KUKA System Software or VW System Software, the model **KUKA smartPAD-2** is used.

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. The drives are only disconnected once the machine has come to a standstill. **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. The drives are only disconnected once the machine has come to a standstill. **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.
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)
VSS	VW System Software
External axis	Motion axis which is not part of the manipulator but is controlled by the robot controller, e.g. KUKA linear unit, turn-tilt table, positioner.

3.2 Personnel

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

- User
- Personnel

Qualification of personnel
Work on the system must only be performed by personnel that is able to assess the tasks to be carried out and detect potential hazards. Death, severe injuries or damage to property may otherwise result. The following qualifications are required:

- Adequate specialist training, knowledge and experience
- Knowledge of the relevant operating or assembly instructions, knowledge of the relevant standards
- 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.
- The user must comply with the regulations relating to personal protective equipment (PPE).

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

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 the risk assessment
- Implementing the required safety functions and safeguards
- Issuing the EC declaration of conformity
- Affixing 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.

3.3 Workspace, safety zone and danger zone

Workspaces are to be restricted to the necessary minimum size.

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

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

There must be no shearing or crushing hazards at the loading and transfer areas.

If there are no physical safeguards present, the requirements for collaborative operation in accordance with EN ISO 10218 must be met.

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: KSS

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 device 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 (internal or external enabling)	Safety stop 2	-

Trigger	T1, T2	AUT, AUT EXT
Internal enabling switch pressed down fully or error	Safety stop 1	-
External enabling switch pressed down fully or error	Safety stop 2	-
E-STOP pressed	Safety stop 1	
Error in safety controller or periphery of the safety controller	Safety stop 0	

* In the KR C5 series, the “pressed down fully” position is not communicated to the robot controller as a specific “panic position” signal for external enabling, but only as “enabling not present”. This triggers a safety stop 2.

3.5 Triggers for stop reactions: VSS

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	EXT
Start key released	STOP 2	-
STOP key pressed	STOP 2	
Drives OFF	STOP 1	
Power switched off via main switch or device 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 (internal or external enabling)	Safety stop 2	-
Internal enabling switch pressed down fully or error	Safety stop 1	-
External enabling switch pressed down fully or error	Safety stop 2	-
E-STOP pressed	Safety stop 1	
Error in safety controller or periphery of the safety controller	Safety stop 0	

* In the KR C5 series, the “pressed down fully” position is not communicated to the robot controller as a specific “panic position” signal for external enabling, but only as “enabling not present”. This triggers a safety stop 2.

3.6 Safety functions

3.6.1 Overview of the safety functions

The following safety functions are always present on the industrial robot:

- Operator safety (= connection for the monitoring of physical safeguards)
- EMERGENCY STOP device on the smartPAD
- External EMERGENCY STOP device
- Enabling device on the smartPAD
- External enabling device
- Velocity monitoring in T1

Depending on the hardware configuration, the following safety functions are present on the industrial robot:

- External safety stop, STOP 1
- External safety stop, STOP 2
- External safe operational stop
- EMERGENCY STOP device (additional local device)
- Peripheral contactor US2

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 conditions, however:

- Each EMERGENCY STOP device is pressed at least once every 12 months.
- Each enabling device is checked at least once every 12 months. (>>> *"Function test" Page 39*)
- Peripheral contactor US2 is checked at least once every 12 months (if used).
- The testing of the safety functions is carried out in accordance with the maintenance table. For maintenance table, see chapter “Maintenance” in the documentation of the hardware components used.

DANGER

Risk of fatal injury due to non-operational safety functions or external safeguards
In the absence of operational safety functions or safeguards, the industrial robot can cause death, severe injuries or damage to property.

- If safety functions or safeguards are dismantled or deactivated, do not operate the industrial robot.

Integrate industrial robot into safety system of the overall system
During system planning, the safety functions of the overall system must be planned and designed. Death, severe injuries or damage to property may otherwise result.

- The industrial robot must be integrated into the safety system of the overall system.

3.6.2 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

3.6.3 “Operator safety” signal: KSS

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

Danger to life and limb due to resumption of automatic operation without adequate checking

Following loss of the “Operator safety” signal, it must not be possible to restart automatic operation by merely closing the safeguard. Otherwise, for example, the safety gate could close unintentionally, thereby causing automatic operation to resume while there are persons in the danger zone. Death, severe injuries or damage to property may result.

- Automatic operation must not be resumed until the safeguard has been closed and the closing has been acknowledged.
- The acknowledgement must be designed in such a way that an actual check of the danger zone can be carried out first. Acknowledgement that is automatically and directly triggered by closure of the safeguard is not permitted.
- If closure is acknowledged by a device (e.g. by the PLC), the system integrator must ensure that the acknowledgement is preceded by an actual check of the danger zone.

3.6.4 “Operator safety” signal: VSS

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.

In Manual Reduced Velocity (T1) mode, operator safety can be overridden with the E2/E22 key.

In Manual High Velocity (T2) mode, operator safety can be overridden with the E2/E22+E7 key.

WARNING

Danger to life and limb due to resumption of automatic operation without adequate checking
Following loss of the “Operator safety” signal, it must not be possible to restart automatic operation by merely closing the safeguard. Otherwise, for example, the safety gate could close unintentionally, thereby causing automatic operation to resume while there are persons in the danger zone. Death, severe injuries or damage to property may result.

- Automatic operation must not be resumed until the safeguard has been closed and the closing has been acknowledged.
- The acknowledgement must be designed in such a way that an actual check of the danger zone can be carried out first. Acknowledgement that is automatically and directly triggered by closure of the safeguard is not permitted.
- If closure is acknowledged by a device (e.g. by the PLC), the system integrator must ensure that the acknowledgement is preceded by an actual check of the danger zone.

3.6.5 EMERGENCY STOP devices

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.

Local EMERGENCY STOP device

The local 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

Danger to life and limb due to tools and equipment without EMERGENCY STOP
If tools and other equipment connected to the robot are not integrated into the EMERGENCY STOP circuit, this can result in death, severe injuries or damage to property.

- Integrate tools and other equipment into the EMERGENCY STOP circuit if they could constitute a potential hazard.

Additional EMERGENCY STOP devices

There must always be at least one additional EMERGENCY STOP device installed: This ensures that there is an EMERGENCY STOP device available, even if the smartPAD has been disconnected.

The additional EMERGENCY STOP device can be an additional local device or an external EMERGENCY STOP device.

External EMERGENCY STOP devices are connected via the customer interface.

3.6.6 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 or device switch of the robot controller

  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.

Take Ethernet safety interface into consideration in risk assessment
Failure to take the Ethernet safety interface into consideration in the risk assessment may result in death, severe injuries or damage to property.

- 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.

WARNING

Danger to life and limb due to smartPAD on controller that has been switched off
If a robot controller is switched off, the EMERGENCY STOP device on the smartPAD is not operational. There is a risk of operational and non-operational EMERGENCY STOP devices becoming interchanged.
Death, severe injuries or damage to property may result.

- Cover smartPADs on controllers that have been switched off or remove them from the system.

3.6.7 Enabling device

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

For robot controllers of the KR C5 series, the model **smartPAD-2** is used. It has 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.

WARNING

Danger to life and limb due to lack of reaction when an enabling switch is released
Releasing one of multiple enabling switches held in the center position does not trigger a stop reaction.
If multiple switches are held in the center position, the robot controller cannot distinguish whether one of them was intentionally released or if it was unintentionally released as the result of an accident.

- Create awareness for the hazard.

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

Danger to life and limb due to manipulation of enabling switches
The enabling switches must not be held down by adhesive tape or other means or tampered with in any other way. Death, severe injuries or damage to property may result.

- Carry out a visual inspection of the enabling switches.
- Rectify tampering or remove any foreign bodies.

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.

3.6.8 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.

3.6.9 Velocity monitoring in T1

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

The axis-specific monitoring can be configured via the parameter **Maximum velocity T1**. 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.

3.7 Additional protective equipment

3.7.1 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.

3.7.2 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.

3.7.3 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

Danger to life and limb following collision with obstacle
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. Death, injuries or damage to property may result.

- Put manipulator out of operation.
- Put external axis out of operation.
- KUKA must be consulted before they are put back into operation.

3.7.4 Mechanical axis limitation (optional)

Some manipulators can be fitted with adjustable 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 mechanical limitations 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 the manufacturer.

3.7.5 Options for moving the manipulator without drive energy

Qualification of personnel with regard to behavior in emergency situations
In emergencies or other exceptional situations, it may be necessary to move the manipulator without drive energy.

- Personnel must be trained in how to move the manipulator without drive energy.

Description

The following options may be 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 can be requested from the manufacturer.

3.7.6 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.

3.7.7 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.

If there are no physical safeguards present, the requirements for collaborative operation in accordance with EN ISO 10218 must be met.

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 hazard 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.

3.7.8 “Drives ready” lamp

If the system is to comply with ANSI/UL 1740, the system integrator must integrate a “Drives ready” lamp into the system. An interface is available for the lamp connection.

The “Drives ready” lamp must always be functionally tested before entering the danger zone. If the test is not passed, the danger zone must not be entered except to work on the lamp.

More information about the “Drives ready” lamp, the associated interface and testing can be found in the assembly instructions for the robot controller.

3.8 Operating mode selection: KSS

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

Danger to life and limb due to mode selector switch without access restriction

If the smartPAD is equipped with a mode selector switch without a key, all persons can operate the mode selector switch, irrespective of their field of activity or qualifications. Death, severe injuries or damage to property may result.

- An additional device must be installed to ensure that the mode selector switch can only be operated by a restricted group of people.
- The device itself must not trigger motions of the industrial robot or other hazards.

3.9 Overview of operating modes and safety functions: KSS

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	-	-	-

Safety functions	T1	T2	AUT	AUT EXT
Jog mode	active	active	-	-
Software limit switches	active	active	active	active
Interface for “Drives ready” lamp	active	active	active	active

3.10 Operating mode selection: VSS

Operating modes

The industrial robot can be operated in the following modes:

- Manual Reduced Velocity (T1)
- Manual High Velocity (T2)
- Automatic External (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
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.

Bypassing

In order to be able to move the manipulator in operating mode T1 or T2 with the safety gate open, the following keys are available:

Active key	Authorization
E2/E22	Authorization to move in T1 with the safety gate open
E2/E22 and E7	Authorization to move in T2 with the safety gate open

WARNING

Danger to life and limb of persons in danger zone in T2
In T2, the robot moves at the programmed velocity. Death, severe injuries or damage to property may result.

- There must be no persons in the danger area.

Use E keys in a safety-conscious manner
Keys E2/E22 and E7 must be used in a safety-conscious manner. Death, severe injuries or damage to property may otherwise result.

- It is the responsibility of the user or system integrator to determine which key may be used and when. The applicable laws, regulations and standards must be taken into consideration.
- The keys may only be used by qualified personnel.

Active key	Safety gate	Jogging possible?		Program execution possible?	
		T1	T2	T1	T2
No key active	Open	No	No	No	No
	Closed	Yes	No	Yes	Yes
E2/E22	Open	Yes	No	Yes	No
	Closed	No	No	No	No
E2/E22 and E7	Open	Yes	No	No	Yes
	Closed	No	No	No	No

3.11 Overview of operating modes and safety functions: VSS

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

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

* In T1 mode, operator safety can be bypassed with the E2/E22 key.

** In T2 mode, operator safety can be bypassed with the E2/E22+E7 key.

3.12 Safety measures

3.12.1 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

Risk of fatal injury due to non-operational safety functions or external safeguards
In the absence of operational safety functions or safeguards, the industrial robot can cause death, severe injuries or damage to property.

- If safety functions or safeguards are dismantled or deactivated, do not operate the industrial robot.

DANGER

Danger to life and limb of persons under the robot arm
Sagging or falling parts can cause death or serious injuries. This applies at all times, e.g. also for assembly tasks or with the controller switched off.

- Never loiter under the robot arm.

WARNING

Danger to life and limb due to incorrect loads
Operating a robot with incorrect loads may result in death, severe injuries or damage to property.

- Use correct load data.
- Use only loads for which the robot is suitable.

CAUTION

Risk of burns from hot motors
The motors reach temperatures during operation which can cause burns.

- Avoid contact.
- Take appropriate safety precautions, e.g. wear protective gloves.

Implants

WARNING

Danger to life due to malfunction of implants caused by motors and brakes
Electric motors and brakes generate electric and magnetic fields. The fields can cause malfunctions in active implants, e.g. pacemakers.

- Affected persons must maintain a minimum distance of 300 mm from motors and brakes. This applies to both energized and deenergized motors and brakes.

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

Danger to life and limb due to disconnected smartPAD
If a smartPAD is disconnected, its EMERGENCY STOP device is not operational. There is a risk of connected and disconnected smartPADs being interchanged. Death, injuries or damage to property may result.

- Remove the disconnected smartPAD from the system immediately.
- Store the disconnected smartPAD out of sight and reach of personnel working on the industrial robot.

The enabling switches on the smartPAD must be subjected to a function test at least once every 12 months and in certain specific cases.
(>>> *"Function test" Page 39*)

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

In the case of faults on the industrial robot, the following safety measures must be implemented immediately:

- 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.

Carry out a functional test after the fault has been rectified.

3.12.2 IT security

KUKA products must only be used in perfect technical condition in accordance with their intended use and only by safety-conscious persons.

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

Take measures to ensure IT security
IT security involves not only aspects of information and data processing as such, but also affects at least the following areas:

- Technology, organization, personnel, infrastructure

KUKA urgently recommends that users 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 cybersecurity authorities

National authorities often make their recommendations available on the Internet.

3.12.3 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.

3.12.4 Start-up and recommissioning: KSS/VSS

KSS only:

Changing default passwords
The system software is supplied with default passwords for the user groups. If the passwords are not changed, this enables unauthorized persons to log on.

- Before start-up, change the passwords for the user groups.
- Only communicate the passwords to authorized personnel.

KSS and VSS:

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.

WARNING

Danger to life and limb due to incorrectly assigned cables
The robot controller is preconfigured for the specific industrial robot. The manipulator and other components can receive incorrect data if they are connected to a different robot controller. Death, severe injuries or damage to property may result.

- Only connect the manipulator to the corresponding robot controller.

WARNING

Risk of fatal injury due to non-configured external axes
The robot controller cannot detect an external axis that is physically connected, but not configured correctly in the software. It cannot exert any torque nor any holding torque on this external axis. If the brakes release, uncontrolled motion can occur at this external axis as a result. Death, severe injuries or damage to property may result.

- Ensure that external axes are correctly configured before an enabling switch is pressed and the brakes are thus released.

Do not impair safety functions
Additional components (e.g. cables and hoses) not supplied by KUKA may be integrated into the industrial robot. If the safety functions are not taken into consideration, this may result in death, severe injuries or damage to property.

- Additional components must not impair or disable safety functions.

NOTICE

Damage to property due to condensation
If the internal cabinet temperature of the robot controller differs greatly from the ambient temperature, condensation can form. This may result in damage to property.

- Wait until the internal cabinet temperature has adapted to the ambient temperature in order to avoid condensation.

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.

WARNING

Danger to life and limb resulting from external forces
The external application of force, such as an impact or a collision, 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, severe injuries or damage to property may result from non-visible damage.

- Check the robot for damage that could have been caused by external forces, e.g. dents or abrasion of paintwork.

  Check the motor and counterbalancing system particularly carefully.

  (Motor inspection not relevant for robots with internal motors.)
- In the case of damage, the affected components must be exchanged.

- There are no foreign bodies or defective 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

3.12.4.1 Checking machine data and safety configuration

WARNING

Danger to life and limb due to incorrect data
The industrial robot must not be moved if incorrect machine data or an incorrect controller configuration are loaded. Unforeseeable reactions may occur. Death, severe injuries or damage to property may result.

- Only operate industrial robots with correct data.

- 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.
- If the activation code of the safety configuration has changed, the safe axis monitoring functions must be checked.

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 for 6-axis robots

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

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.

General practical test 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.

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

First part:

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. Mark the starting position of the TCP.

   Also read and note the starting position from the **Actual position** – **Cartesian** display on the smartHMI.
3. 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.
4. Measure the distance covered and compare it with the distance value displayed on the smartHMI. The deviation must be < 5%.
5. 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.

General practical test for SCARA robots

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

First part:

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 on A4.
2. Mark the starting position of the TCP.
   Also read and note the starting position from the **Actual position** – **Cartesian** display on the smartHMI.
3. Jog the TCP in the Z direction. The distance must be at least 20% of the maximum travel. Determine the exact length via the **Actual position** display.
4. Measure the distance covered and compare it with the distance value displayed on the smartHMI. The deviation must be < 5%.

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

Second part:

- Rotate the tool manually 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 ROBROOT kinematic system

If practical tests are required for the machine data, this test must be carried out if the robot is mounted on a mathematically coupled ROBROOT kinematic system, e.g. on a KL.

- Move the axes of the ROBROOT kinematic system manually in Cartesian mode, individually and one after the other.
  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.

3.12.4.2 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**
  Start-up mode is always possible.
- **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 motion enable is present, the US2 voltage is switched on – even in Start-up mode.



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

Risk of fatal injury due to non-operational external safeguards
Use of Start-up mode disables all external safeguards. Death, severe injuries or damage to property may result.

- There must be no persons in the danger zone of the manipulator while it is in Start-up mode.

Misuse

Any use or application deviating from the intended use is deemed to be misuse and is not allowed. It will result in the loss of warranty and liability claims. KUKA is not liable for any damage resulting from such misuse.

3.12.5 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 jam 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 can be avoided, there must be no persons inside the safeguarded area.

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 more than one person 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 zone 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 the 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.

3.12.6 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.

3.12.7 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 or the requirements for collaborative operation in accordance with EN ISO 10218 have been met.
- 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.

3.12.8 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

Danger to life and limb due to live parts
The robot system must be disconnected from the mains power supply prior to work on live parts. It is not sufficient to trigger an EMERGENCY STOP or safety stop, because parts remain live. Death or severe injuries may result.

- Before commencing work on live parts, turn off the main switch and secure it against being switched on again.

  If the controller variant in question does not have a main switch (e.g. KR C5 micro), turn off the device switch then disconnect the power cable and secure it so it cannot be reconnected.
- Then check to ensure that the system is deenergized.
- Inform the individuals involved that the robot controller is switched off. (e.g. by affixing a warning sign)

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.

On robot controllers with transformers, the transformers must be disconnected before working on components in the robot controller.

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.

- **Counterbalancing system classified below category I**: Is subject to the Pressure Equipment Directive but exempt from application of the Pressure Equipment Directive according to Art. 4, para. 3 and therefore not CE marked.
- **Counterbalancing system classified as category I or higher**: Is subject to the Pressure Equipment Directive and CE marked as a component (see rating plate of the counterbalancing system). The pressure equipment is placed on the market in conjunction with partly completed machinery. Conformity is expressed on the declaration of incorporation according to the Machinery Directive.

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

- In Germany, the counterbalancing system is work equipment according to the German Ordinance on Industrial Safety and Health (BetrSichV). Inspection intervals in Germany in accordance with the Ordinance on Industrial Safety and Health, Sections 14 and 15. Inspection by the user before commissioning at the installation site.
- Inspection intervals in all other countries must be researched and observed. As a rule, however, at least the maintenance intervals specified by KUKA must be observed. These must not be exceeded.

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

- The 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.

Use current safety data sheets
Knowledge of the safety data sheets of the substances and mixtures used is a prerequisite for the safe use of KUKA products. Death, injuries or damage to property may otherwise result.

- Request up-to-date safety data sheets from the manufacturers of hazardous substances regularly.

3.12.9 Decommissioning, storage and disposal

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

3.12.10 Safety measures for single point of control

Overview

If certain components are used on the industrial robot, 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
- KUKA.DeviceConnector
  (not KUKA.DeviceConnector pre-installed)

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.

Further safety measures for Single Point of Control
Depending on the specific application, further safety measures may be required to ensure complete implementation of the principle of “single point of control”. Failure to take this precaution into consideration may result in death, injuries or damage to property.

- Check whether further safety measures are required; if so, implement them.

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.

- In T1 and T2, the system variable $OV_PRO must not be written to by the submit interpreter or the PLC.

OPC server, KUKA.DeviceConnector, 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.

4 Operation

4.1 smartPAD

Description

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.

smartPAD-2

For robot controllers of the KR C5 series, the model **smartPAD-2** is used. This model has a number of variants, e.g. with different lengths of connecting cables.

In this documentation, the designation “KUKA smartPAD” or “smartPAD” refers to the model **smartPAD-2**.

4.1.1 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.

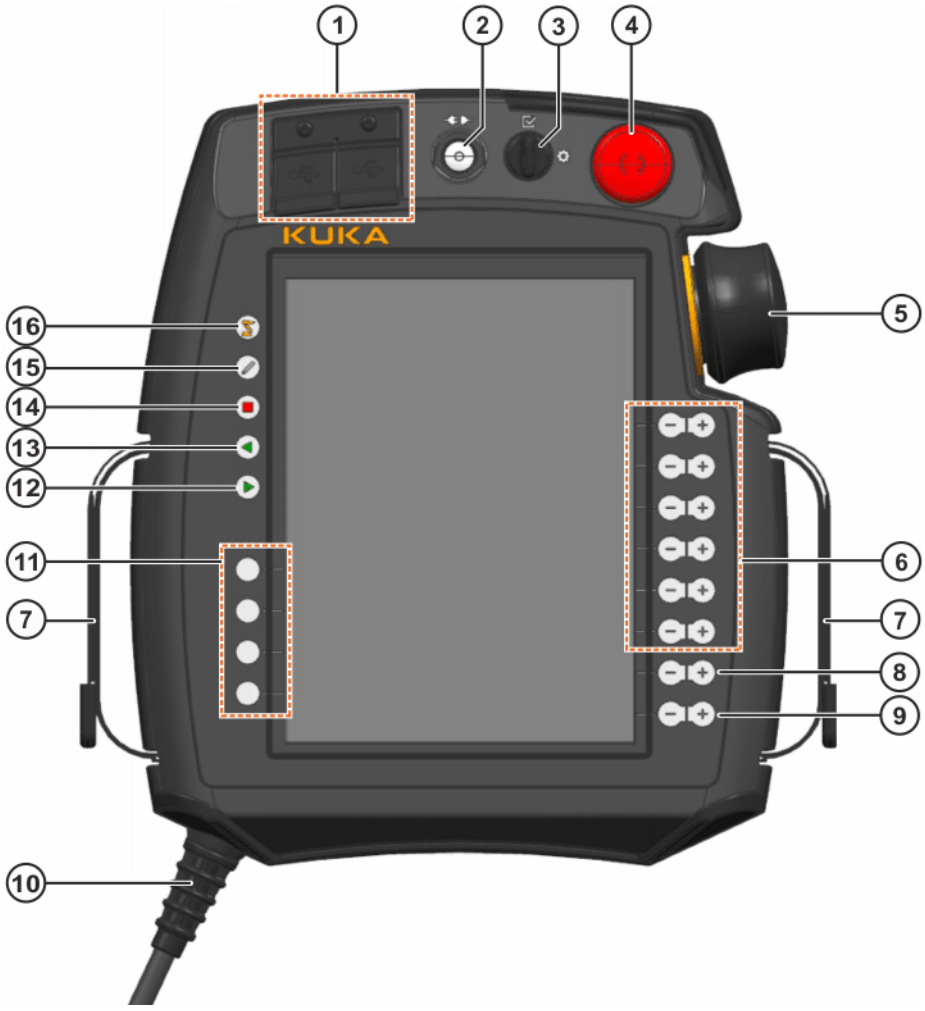


Fig. 4-1: Front of smartPAD-2

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
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. (>>> *4.15 "Changing operating mode" Page 84*)
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 67*)
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 Rear of smartPAD-2

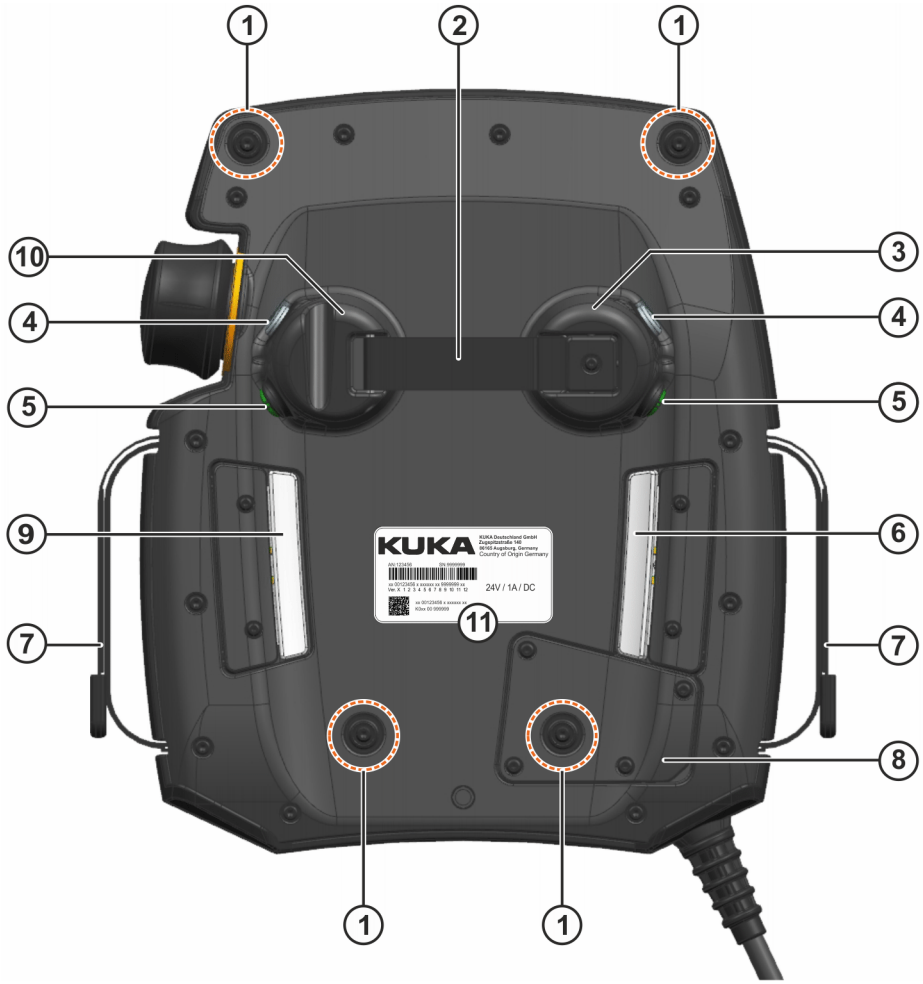


Fig. 4-2: Rear of smartPAD-2

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

WARNING

Risk of fatal injury due to non-operational EMERGENCY STOP device
If the smartPAD is disconnected, the system can no longer be switched off by means of the EMERGENCY STOP device on the smartPAD. Measures must be taken to prevent operational and non-operational EMERGENCY STOP devices from being mixed up.
Death, injuries or damage to property may result.

- Connect an external EMERGENCY STOP to the robot controller.
- Remove the disconnected smartPAD from the system immediately.

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 39*)
   - Check whether the smartHMI is displayed again. (This may take longer than 30 s.)

WARNING

Risk of fatal injury due to non-operational EMERGENCY STOP device
If a non-operational smartPAD remains connected, there is the danger that the user will attempt to activate a non-operational EMERGENCY STOP. Death, injuries or damage to property may result.

- Disconnect a non-operational smartPAD and remove it from the system immediately.

4.2 KUKA smartHMI user interface

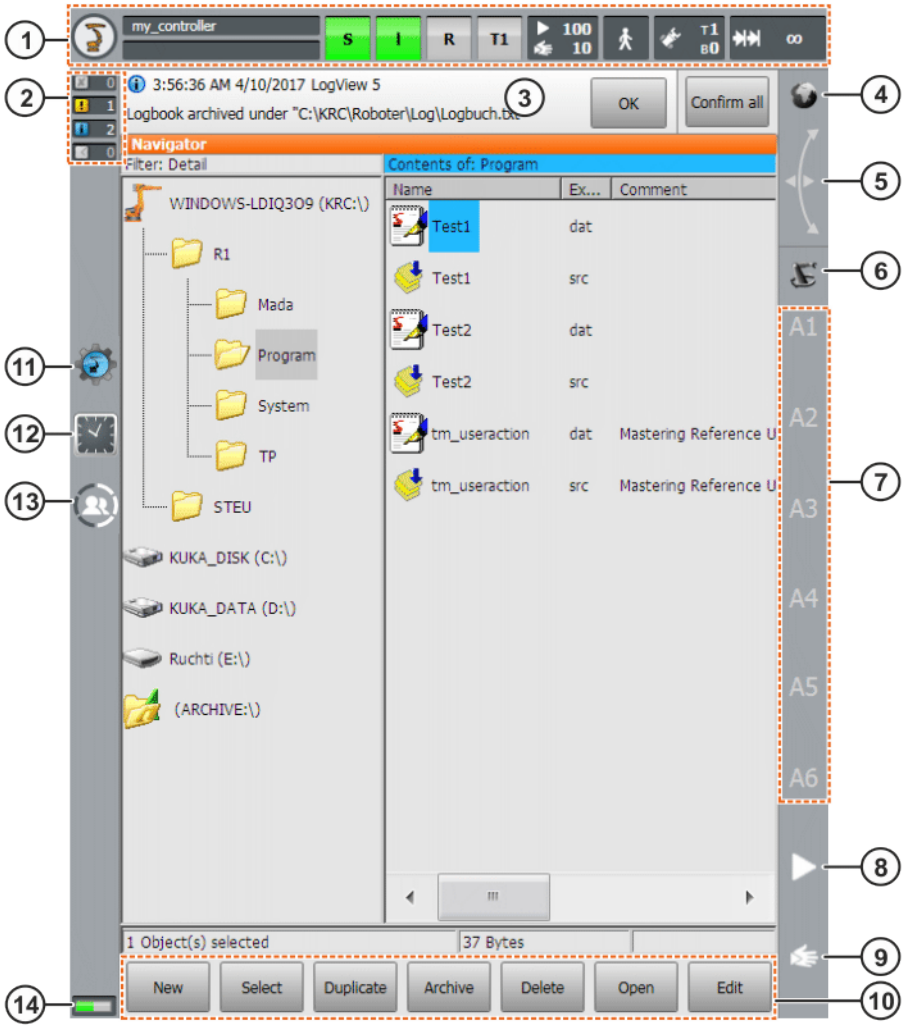


Fig. 4-3: KUKA smartHMI user interface

Item	Description
1	Status bar (>>> *4.2.2 "Status bar" Page 67*)
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 reference system for jogging with the Space Mouse. Touching the indicator displays all reference systems, allowing a different one to be selected. Required user rights: Function group **General jog settings**
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. (>>> *4.17.7 "Defining the alignment of the Space Mouse" Page 96*)
6	**Jog keys** status indicator This indicator shows the current reference system for jogging with the jog keys. Touching the indicator displays all reference 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 reference system axes are displayed here (X, Y, Z, A, B, C). Touching the label indicates which kinematics group is selected.
8	Program override (>>> *8.4 "Setting the program override" Page 381*)
9	Jog override (>>> *4.17.2 "Setting the jog override" Page 92*)
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.

Item	Description
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. (>>> *4.13 "Changing user group" Page 82*)
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.

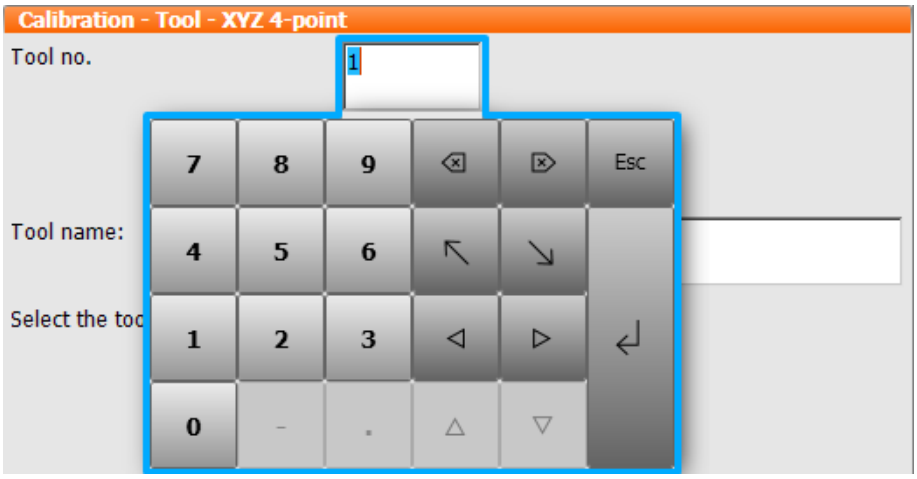


Fig. 4-4: Example keypad

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

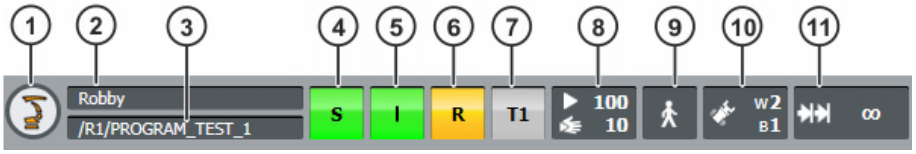


Fig. 4-5: KUKA smartHMI status bar

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 692*)
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 68*)
6	**Robot interpreter** status indicator. Programs can be reset or canceled here. (>>> *8.5 "Robot interpreter status indicator" Page 382*)
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.
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.

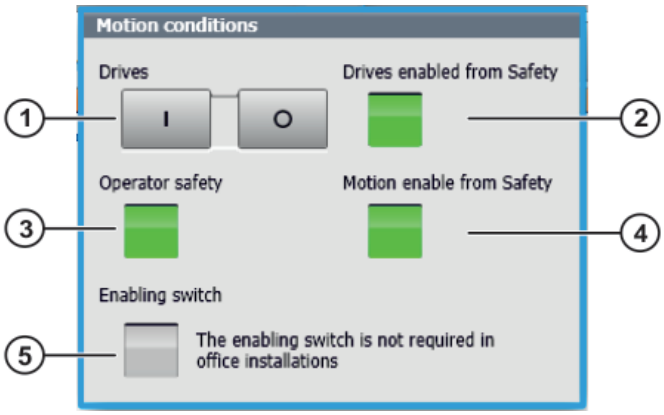


Fig. 4-6: “Motion conditions” window

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 70*)
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:

KSS/VSS	Operating mode	Condition
KSS	T1, T2	• The enabling switch is pressed.
	AUT, AUT EXT	• The physical safeguard is closed.
VSS	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 **smartHMI** 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.

Carrying out specific steps after switching on for the first time
If the robot controller is 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 137*)

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.

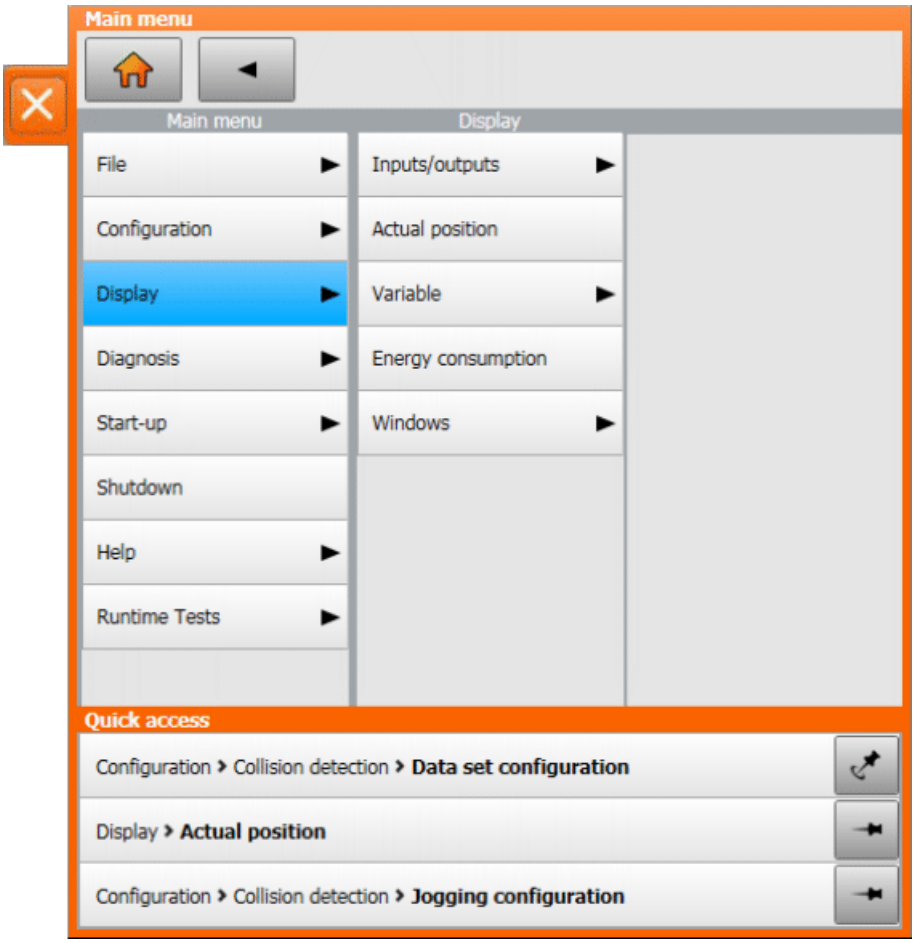


Fig. 4-7: Example: The Display submenu is open.

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 76*)
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 73*)

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

Damage to system files by actuating the main switch too soon
If the **Reboot control PC** option is selected and the main switch is actuated before the restart is completed, system files can be damaged.

- Wait until the restart is completed. Only then actuate the main switch.

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

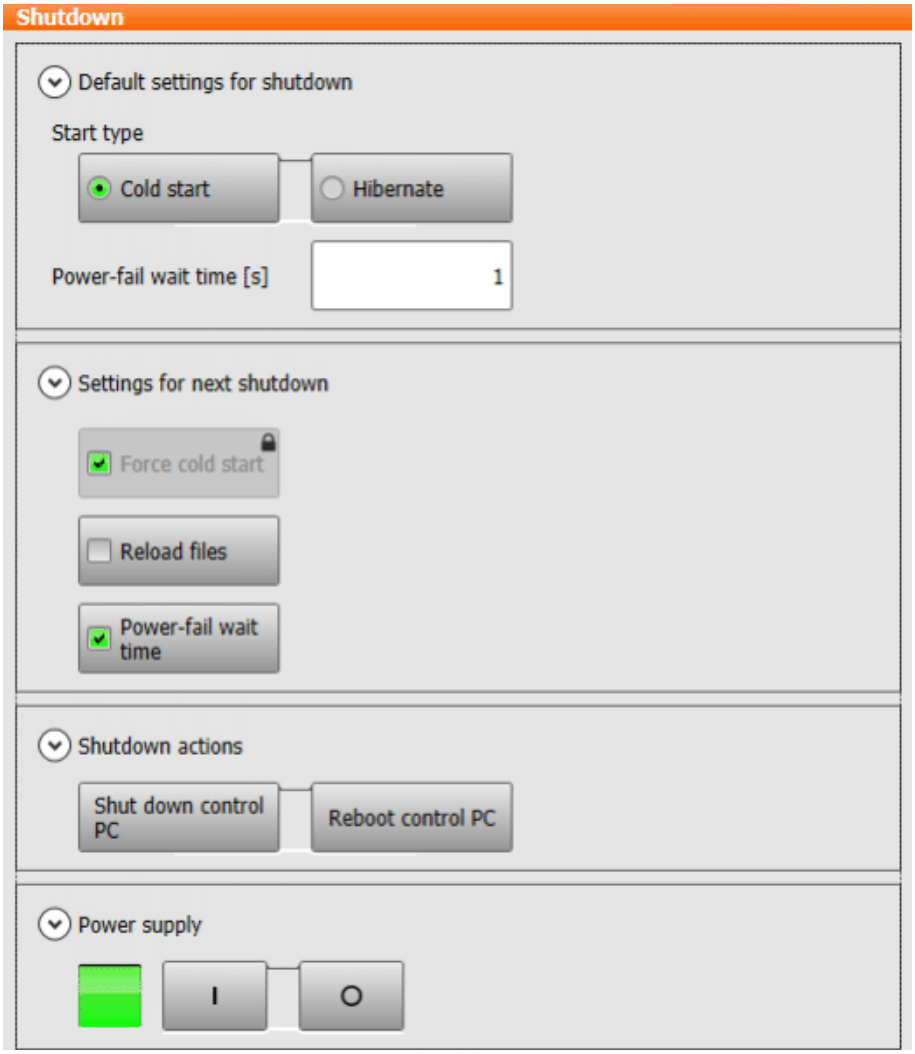


Fig. 4-8: “Shutdown” window

Option	Description
Default settings for shutdown Required user rights: Function group **Critical configurations**	
Cold start	**Hibernate** is the default start type. (>>> *"Start types" Page 76*)
Hibernate	

Option	Description
Power-fail wait time	In the case of a power failure, the robot comes to a standstill. The robot controller does not shut down immediately, but only after the power-fail wait time. In other words, brief power failures can be overridden through this wait time. The error messages must then only be acknowledged and the program can be resumed. • 0 … 1,000 s Further information: • Switching off via the main switch also represents a power failure. • The smartPAD is turned off in the event of a power failure. If the voltage returns within the power-fail wait time, it is automatically turned back on. • During the wait time, the robot controller is powered by its battery. • During the wait time, the robot controller is in “Bus Power Off mode”. If the power failure lasts longer than the power-fail wait time, then the default start type defined in the **Shutdown** window applies for the restart. **Note**: The LEDs on the CSP indicate the operating state of the robot controller. They also indicate, for example, the Bus Power Off mode. Further information about the LEDs is contained in the assembly instructions of the robot controller.
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-fail wait time	Active: The wait time is adhered to the next time the system is shut down. Inactive: The wait 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.

Option	Description
Power supply Required user rights: Function group **General configuration**	
I / **O**	For (short) phases in which the robot does not move, the power can be switched off via the **O** key to save energy. (>>> *4.7 "Switching off the power to save energy" Page 76*) Status LED: • Green: Power is on. • Red: Power is off. • Gray: State 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 off the power to save energy

Description

For (short) phases in which the robot does not move, the power can be switched off to save energy.

- Here, all components of the robot controller besides the motherboard are switched off.
- The discrete inputs and outputs are not functional.
- The smartPAD is also switched off.

This switch-off mode saves less energy than Hibernate. On the other hand, the controller is active again after a few seconds when switching back on, which is faster than after a Hibernate.

During the time it is switched off, the robot controller is in “Bus Power Off mode”.



The LEDs on the CSP indicate the operating state of the robot controller. They also indicate, for example, the Bus Power Off mode. Further information about the LEDs is contained in the assembly instructions of the robot controller.

Precondition

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

Procedure

1. Select the menu item **Shutdown** in the main menu.
   The **Shutdown** window opens.
2. Press the **O** key under **Power supply**.
   The following message is displayed:
   Caution! Clicking on OK switches off the smartPAD as well! Press the soft power button on the controller to turn the power back on.
3. Answer **OK** to the question to switch off the components.
   (Selecting **Cancel** as the answer means that the power supply continues.)
4. To switch the components back on again later, press the soft power button on the robot controller.

4.8 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 68*)
2. Switch the drives on or off.

4.9 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 failure 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 be resumed.

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

NOTICE

Damage to system files by actuating the main switch too soon
If the **Reboot control PC** option is selected and the main switch is actuated before the restart is completed, system files can be damaged.

- Wait until the restart is completed. Only then actuate the main switch.

Procedure

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

4.10 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**.

Languages

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
	Vietnamese

4.11 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.

4.12 Online documentation and help for messages

4.12.1 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

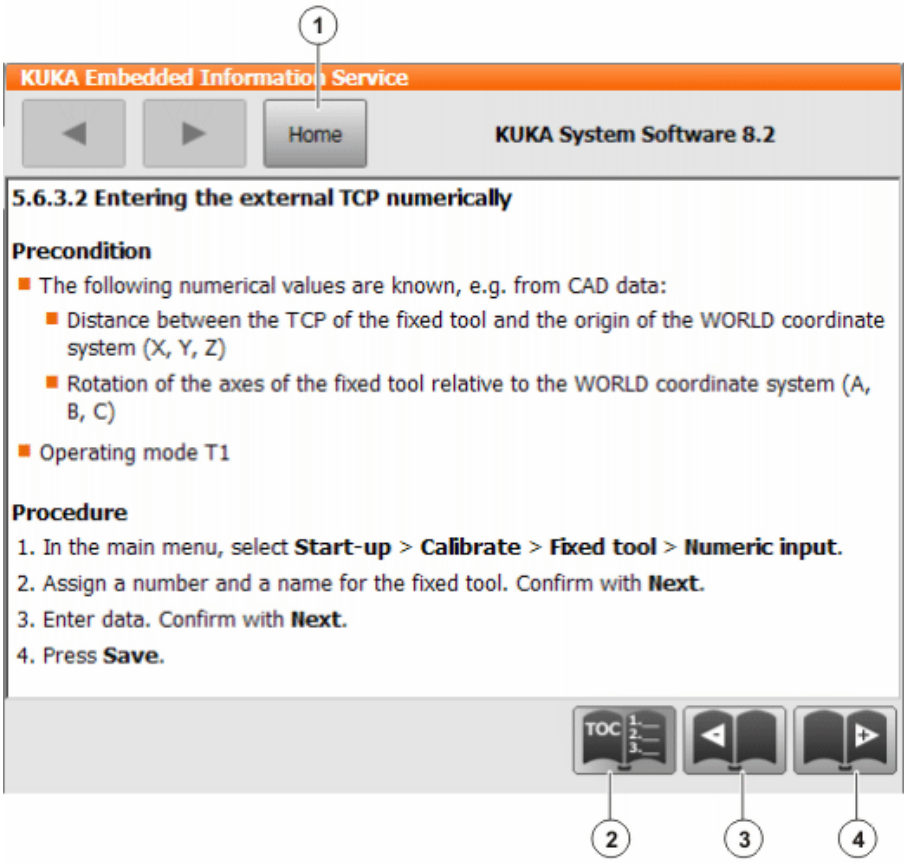


Fig. 4-9: Online documentation – Example from the KUKA System Software

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

4.12.2 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. 4-10*)

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. 4-11*)
   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. 4-10*)
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. 4-11*)
   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

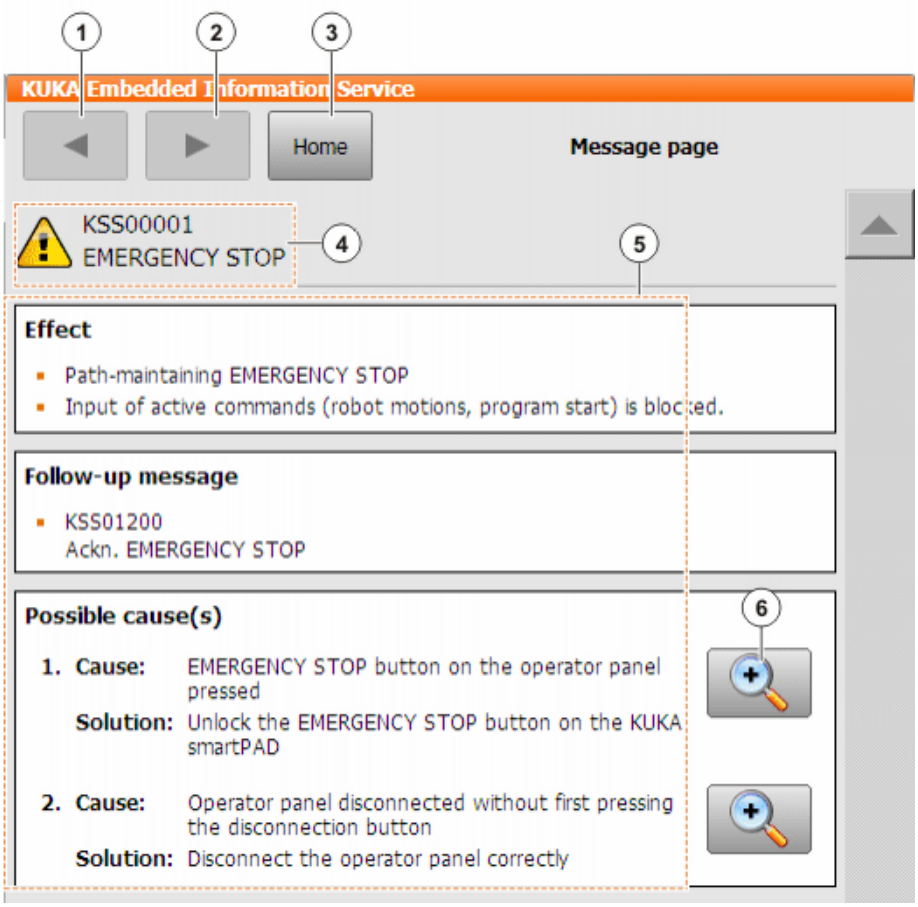


Fig. 4-10: Message page – Example from the KUKA System Software

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. 4-11*)

Detail page

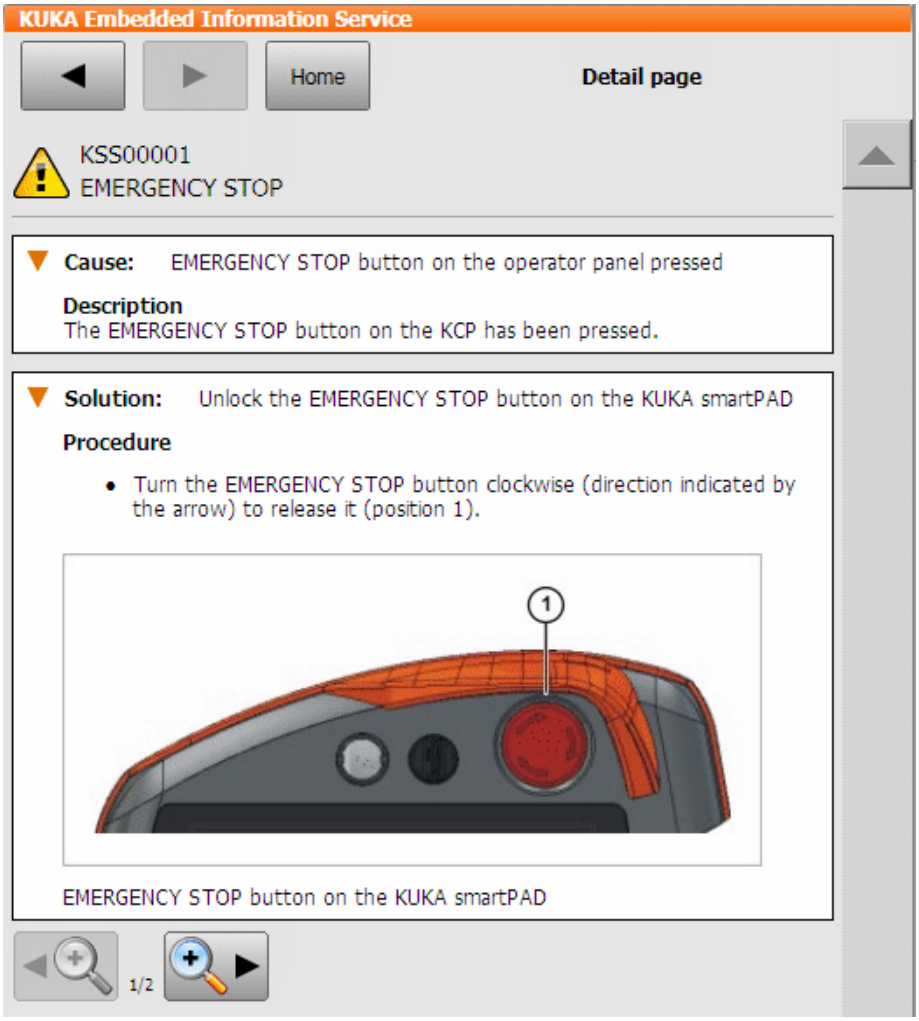


Fig. 4-11: Detail page – Example from the KUKA System Software

4.13 Changing user group

Procedure

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

     Fig. 4-12: 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**.

4.14 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.

(>>> *6.11 "Changing the password" Page 251*)

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.

(>>> *6.12 "Displaying and modifying user rights" Page 251*)

4.15 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. 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

4.16 Coordinate systems

Overview

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

- WORLD
- ROBROOT
- BASE
- TOOL

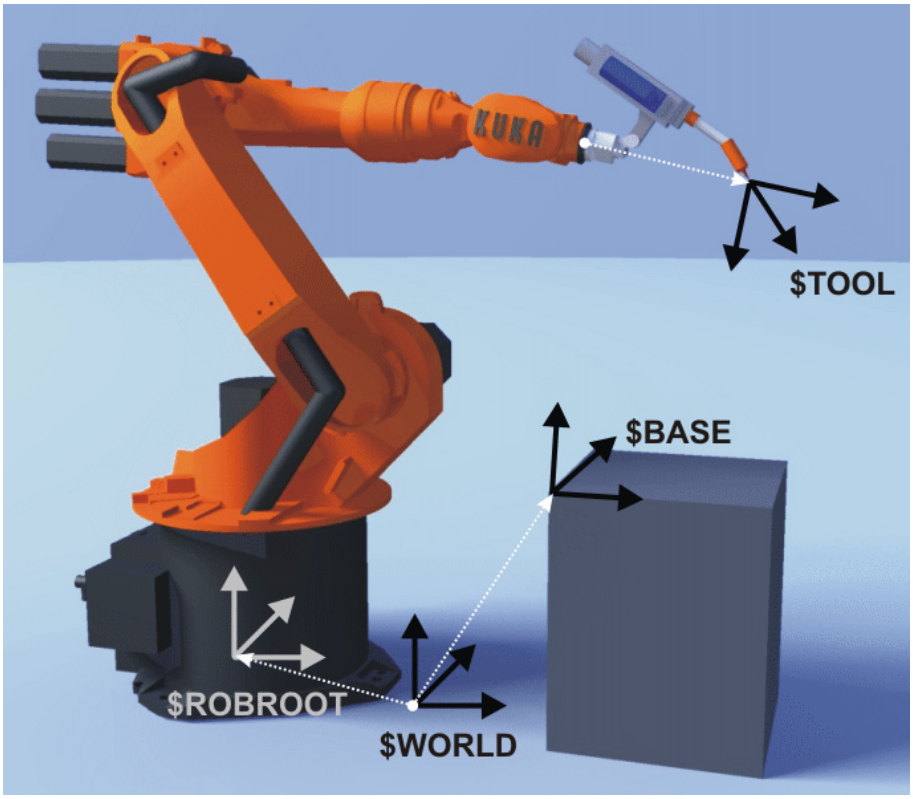


Fig. 4-13: Overview of coordinate systems

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.

(>>> *5.18.4 "Introduction to BASE calibration" Page 192*)

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.

(>>> *5.18.2 "Introduction to TOOL calibration" Page 190*)

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

4.17 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.

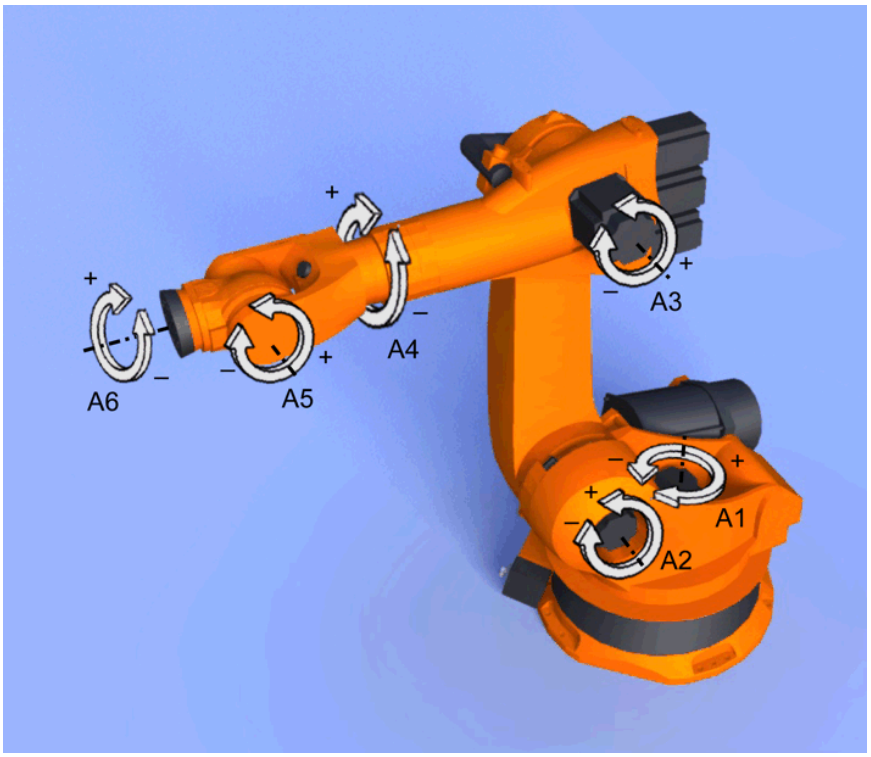


Fig. 4-14: Axis-specific jogging

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	(>>> *4.17.5 "Cartesian jogging with the jog keys" Page 94*)	(>>> *4.17.4 "Axis-specific jogging with the jog keys" Page 93*)
Space Mouse	(>>> *4.17.8 "Cartesian jogging with the Space Mouse" Page 97*)	Axis-specific jogging with the Space Mouse is possible, but is not recommended.

4.17.1 Setting the jogging options

Description

All parameters for jogging the robot can be set in the **Jog 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.

4.17.1.1 “General” tab

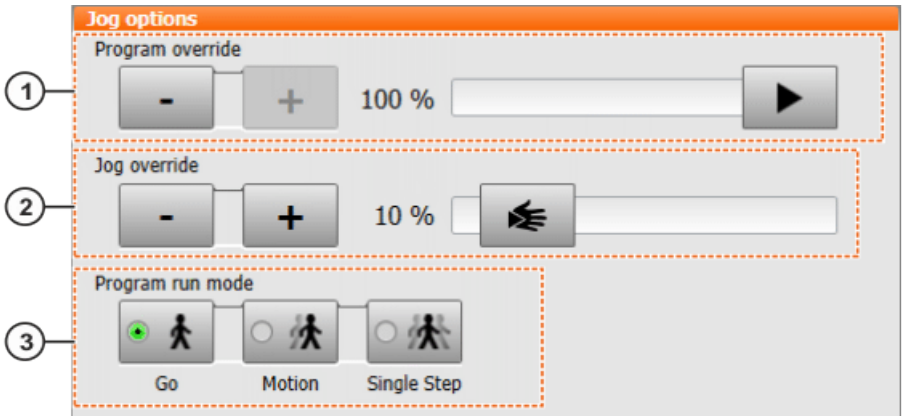


Fig. 4-15: General tab

Item	Description
1	Set the program override. (>>> *8.4 "Setting the program override" Page 381*)
2	Set the jog override. (>>> *4.17.2 "Setting the jog override" Page 92*)
3	Select the program run mode. (>>> *8.1.1 "Program run modes" Page 377*)

4.17.1.2 “Keys” tab

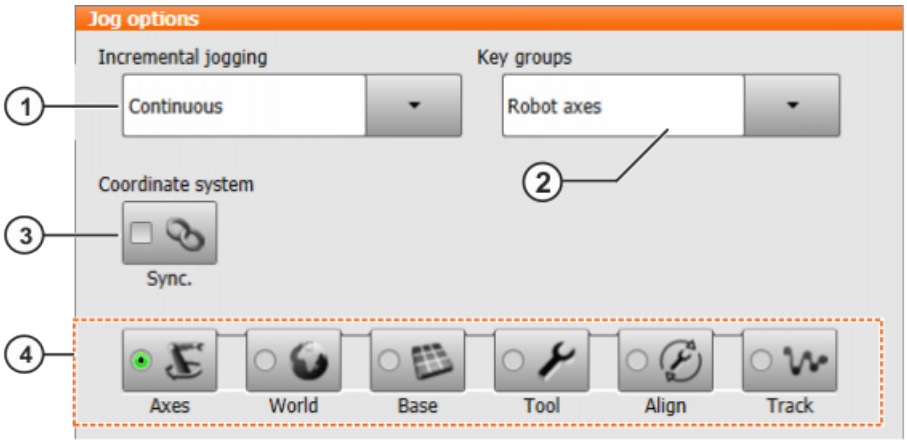


Fig. 4-16: “Keys” tab

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

Item	Description
1	Incremental jogging (>>> *4.17.10 "Incremental jogging" Page 98*) If the option **Track** is selected, the setting for incremental jogging automatically changes to **Continuous** (if not already set). 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. (>>> *4.18 "Jogging external axes" Page 104*) • If the option **Align** is selected, the kinematics group cannot be changed. • If the option **Track** is selected, the kinematics group automatically changes to **No selection**. If **Track** is deselected again, the setting changes back to the original value.

Item	Description
3	Check box **Sync.** • Check box not active (default): on the **Keys** and **Mouse** tabs, different reference systems can be selected (**Axes**, **World**, **Base** or **Tool**). • Check box active: if the reference system on the **Keys** tab is changed, the setting on the **Mouse** tab is adapted automatically and vice versa.
4	• Select the reference system for jogging with the jog keys: **Axes**, **World**, **Base** or **Tool** • **Align**: For simple alignment of the tool with a base (>>> *4.17.11 "Aligning the tool with a base" Page 99*) **Align** is not available for palletizing robots. • **Track**: To execute the most recent motions backwards (>>> *4.17.12 "Backward motion using the jog keys" Page 101*)

4.17.1.3 Mouse tab

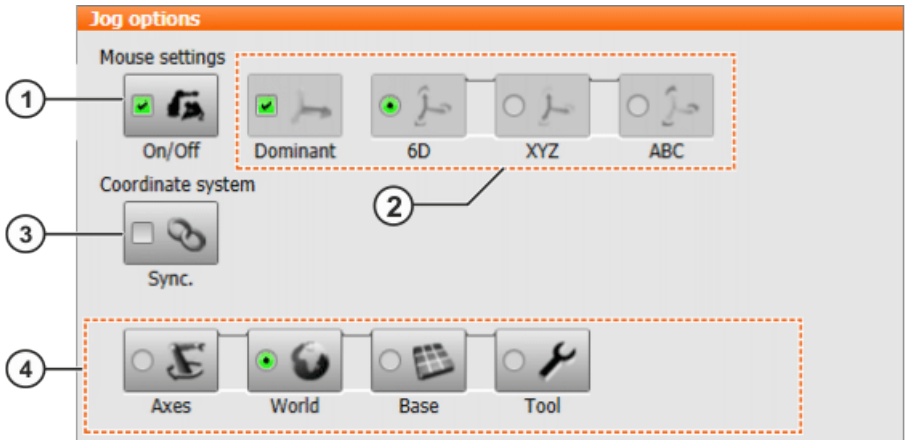


Fig. 4-17: “Mouse” tab

Item	Description
1	• Check box active (default): Space Mouse is active. • Check box not active: Space Mouse is inactive. In the case of a user group change, the Space Mouse is automatically set back to the default state, i.e. “active”. Required user rights: Function group **General jog settings**
2	Configure the Space Mouse (>>> *4.17.6 "Configuring the Space Mouse" Page 94*) Required user rights: Function group **General configuration**

Item	Description
3	Check box **Sync.**: • Check box not active (default): On the **Keys** and **Mouse** tabs, different reference systems can be selected. • Check box active: If the reference system on the **Keys** tab is changed, the setting on the **Mouse** tab is adapted automatically and vice versa. Required user rights: Function group **General jog settings**
4	Reference system for jogging with the Space Mouse Required user rights: Function group **General jog settings**

4.17.1.4 “KCP pos.” tab

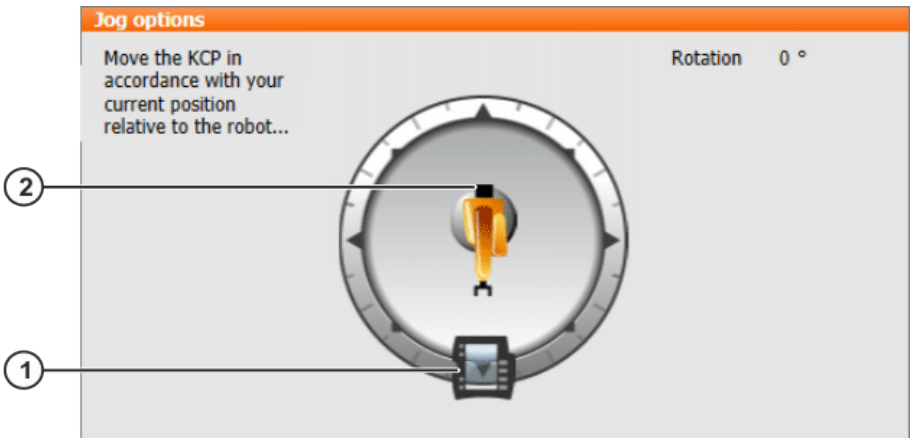


Fig. 4-18: “KCP pos.” tab

Item	Description
1	Drag the smartPAD icon to the position corresponding to the location of the user relative to the junction box. (>>> *4.17.7 "Defining the alignment of the Space Mouse" Page 96*)
2	Reference point: Junction box on the base frame

4.17.1.5 Cur. tool/base tab

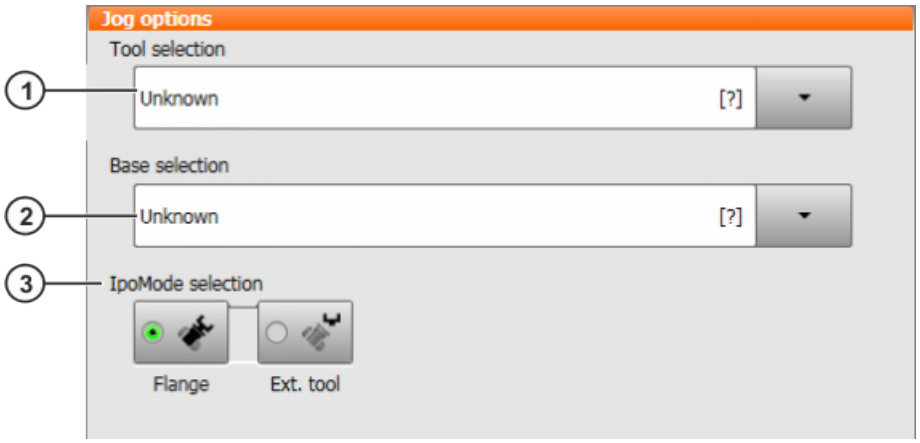


Fig. 4-19: Cur. tool/base tab

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

Item	Description
1	The current tool is displayed here. A different tool can be selected. (>>> *4.17.3 "Selecting the tool and base" Page 93*) The display **Unknown [?]** means that no tool has yet been calibrated.
2	The current base is displayed here. A different base can be selected. (>>> *4.17.3 "Selecting the tool and base" Page 93*) 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.

4.17.1.6 “Collision detection” tab

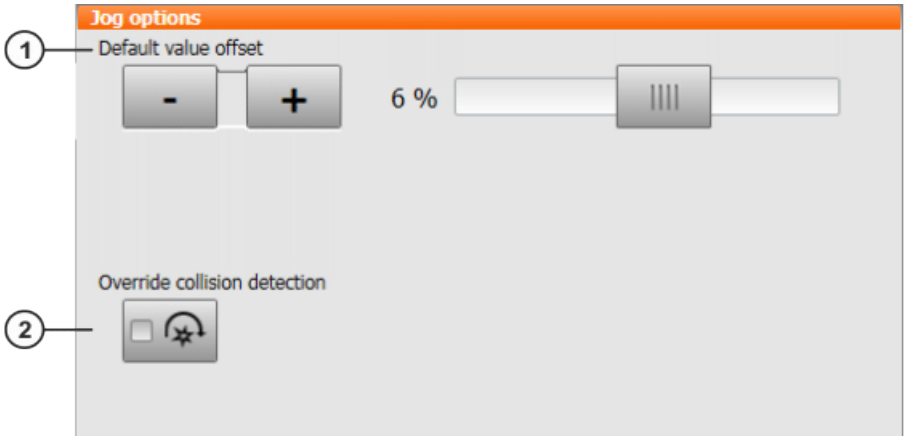


Fig. 4-20: Collision detection tab

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. As long as the setting here does not equal zero, the following status message is displayed: *Collision detection: Jogging values modified by {percentage value}*. The percentage value refers to the values under **Configuration** > **Collision detection** > **Jogging configuration** in the **Data set values** column. • Value 0%: No change compared with **Data set 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 **Track** jog mode. **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.

4.17.2 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. 4-21: 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%.

4.17.3 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. 4-22: 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.

4.17.4 Axis-specific jogging with the jog keys

Precondition

- User rights: function group **Jogging with the jog keys**
- **Axes** is selected as the reference 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.

4.17.5 Cartesian jogging with the jog keys

Precondition

- User rights: function group **Jogging with the jog keys**
- **World**, **Base** or **Tool** is selected as the reference 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.

4.17.6 Configuring the Space Mouse

Precondition

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

Procedure

1. Open the **Jog options** window and select the **Mouse** tab.
   (>>> *4.17.1 "Setting the jogging options" Page 87*)
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

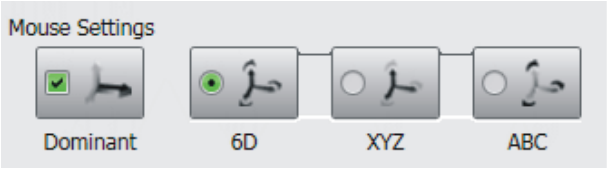


Fig. 4-23: Mouse settings

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

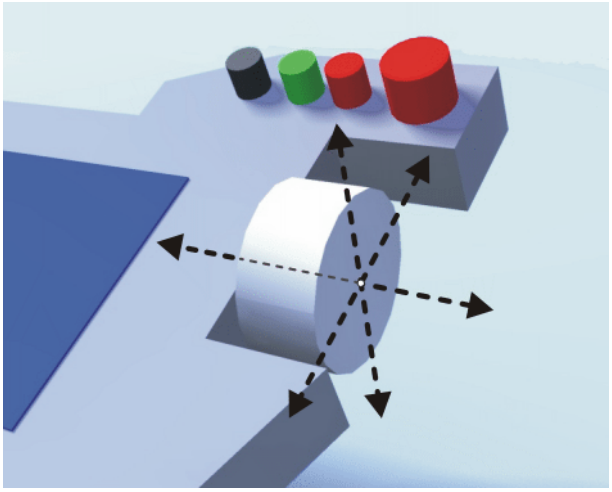


Fig. 4-24: Pushing and pulling the Space Mouse

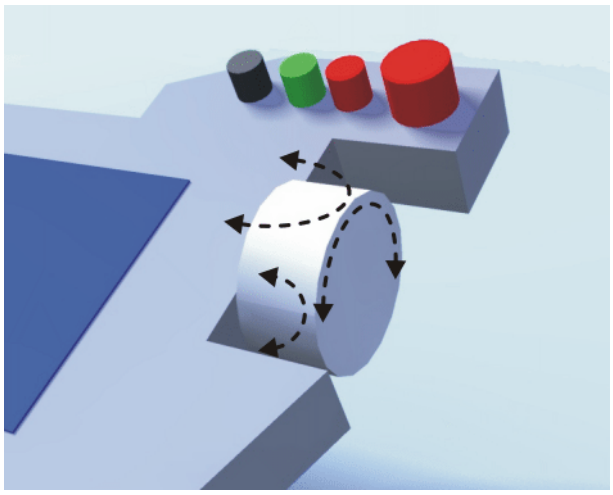


Fig. 4-25: Rotating and tilting the Space Mouse

4.17.7 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°.

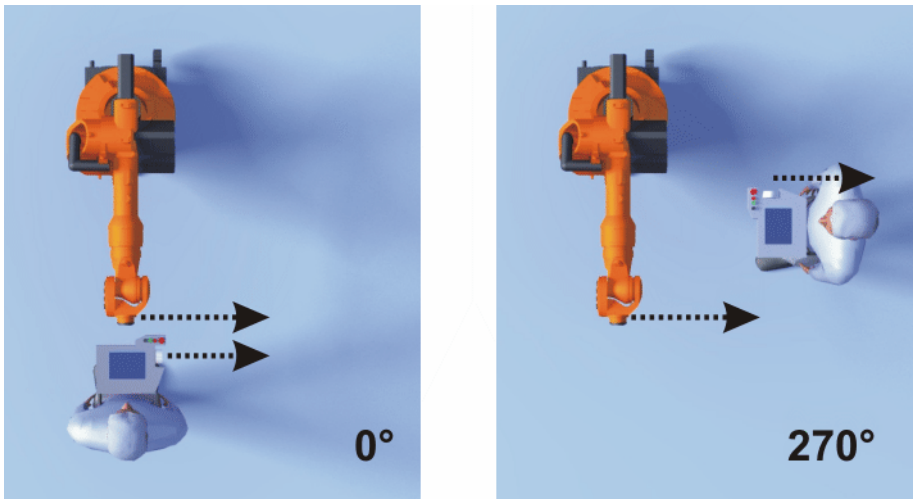


Fig. 4-26: 6D mouse: 0° and 270°

Precondition

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

Procedure

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

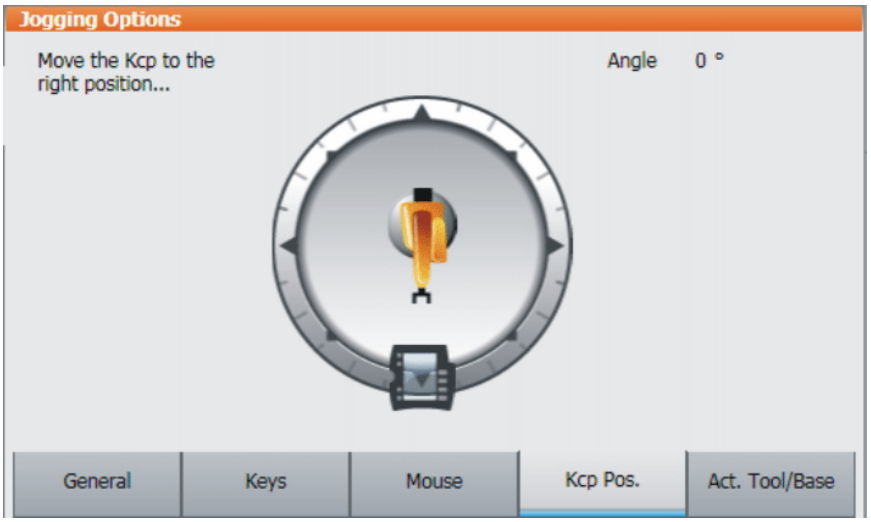


Fig. 4-27: Defining the alignment of the Space Mouse

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

4.17.8 Cartesian jogging with the Space Mouse

Precondition

- User rights: function group **Jogging using the 6D mouse**
- **World**, **Base** or **Tool** is selected as the reference 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.

4.17.9 Temporarily deactivating the Space Mouse

Precondition

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

Procedure

1. Open the **Jog options** window and select the **Mouse** tab.
   (>>> *4.17.1 "Setting the jogging options" Page 87*)
2. In the **Mouse settings** group, remove the check mark for **On/Off**.
   The following message is displayed: *The mouse was temporarily deactivated by the user.*

3. Close the **Jog options** window.
   The Space Mouse is now inactive.

In the case of a user group change, the Space Mouse is automatically set back to the default state, i.e. “active”.

4.17.10 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

Settings/increment sizes:

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

4.17.11 Aligning the tool with a base

Description

The **Align** function makes it possible to align the tool easily with the current base.

- The tool can be aligned perpendicular to an axis of the base. The axis can be selected.
- The tool can be aligned parallel to the selected axis.
- The motion between “perpendicular” and “parallel” can be continuous or incremental.

The tool can be aligned using the jog keys, but not with the Space Mouse.

Details

- $TOOL_DIRECTION is always regarded as the tool direction for the **Align** function. It is thus explicitly independent of $TOOL_DIRECTION_LIN_CIRC.
- **Align** can also be used for fixed tools. In this case, $TOOL_DIRECTION refers to the fixed tool. The selected axis refers to the workpiece (mounted on the robot).
- If no coordinate system is defined for the tool or base, or if TOOL or BASE is invalid, $NULLFRAME is used.

Precondition

- User rights of the following function groups:
  - **General jog settings**
  - **Jogging with the jog keys**
- The desired jog override is set.
- Tool and base have been selected.
- T1 mode

Procedure

1. Move the TCP to the workpiece. Move roughly to the position from which alignment of the tool is to be carried out.

   Finer positioning can be carried out subsequently using the special jog keys.
2. If motion is to be incremented: select the desired increment in the status bar.
3. For jogging, select the option **Align** on the **Keys** tab.

   (>>> *4.17.1.2 ""Keys" tab" Page 88*)

   The corresponding icons are now displayed next to the jog keys on the smartHMI.
4. Using the keys **1+2**, select the axis to which the tool is to be aligned.
5. If required, carry out finer positioning of the tool now (or after alignment), e.g. move it forward or backward in the tool direction using key **3**.

6. Align the tool using key **4**.
7. The motion extends over a maximum of 180°. As soon as the center of the maximum possible overall range is reached (i.e. 90°), the robot stops there automatically.

   The tool direction is now perpendicular to the set axis.

   The motion can be resumed by pressing the jog key again.
8. Once the end of the motion range is reached, the following message is displayed: *Motion to coordinate axis {Selected axis} finished*.

   The tool direction is now parallel to the set axis.

Jog keys

The tool is aligned with the base using key **4**.

The other keys are assigned other jog options.

Key	Icon	Description
1+2	Example: 	**Setting the plane and axis for alignment** In the example on the left, the XY plane of the base is set and thus the Z axis. **Important:** The set axis is always the one perpendicular to the plane, i.e. the axis that is not (!) indicated by an icon. In the example on the left it is thus the Z axis. (>>> *"Setting a different plane/axis" Page 101*)
		Moving along the workpiece When the jog keys are pressed, the TCP moves in the set plane of the base (in the example: XY plane).
3		**Move in the tool direction** (change the distance from the workpiece) • **Plus**: In the tool direction towards the workpiece • **Minus**: In the tool direction away from the workpiece
4		**Align the tool with the base** (set the angle to the workpiece) Decisive here is the plane or axis set using keys **1+2** (in the example above for keys **1+2**: XY plane / Z axis). • **Plus**: raises the tool direction towards the axis At the end of the motion range, the tool direction is parallel to the axis. The plus tool direction then corresponds to the minus direction of the selected axis. • **Minus**: lowers the tool direction towards the axis At the end of the motion range, the tool direction is parallel to the axis. The plus tool direction then corresponds to the plus direction of the selected axis. The motion extends over a maximum of 180°. As soon as the center of the maximum possible overall range is reached (i.e. 90°), the robot stops there automatically. The tool direction is now perpendicular to the set axis. The motion can be resumed by pressing the jog key again.
5		**Rotate tool** The tool rotates about the tool direction. • **Plus**: Clockwise • **Minus**: Counterclockwise

Key	Icon	Description
6		**Swivel tool** The tool swivels about the set axis. The angle relative to the workpiece is not changed. • **Plus**: Clockwise • **Minus**: Counterclockwise

Setting a different plane/axis

1. Touch any icon on the jog key bar. A view opens with the 3 possible planes.
2. Touch the desired plane in the view.
   The view closes. The icon indicates the selected plane.

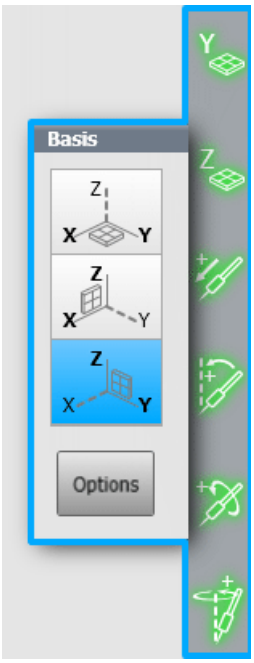


Fig. 4-28: View with 3 possible planes

In the example, the YZ plane is selected and thus the X axis.

4.17.12 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: (>>> *8.12 "Backward motion using the Start backwards key" Page 386*) An overview of the most important differences can be found here: (>>> *8.12.4 "Comparison of "Start backwards"/backwards using the jog keys" Page 391*)

4.17.12.1 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 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.

4.17.12.2 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 of the smartPAD.

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 using the **Track** option and then starts another motion, part of the buffer is deleted.

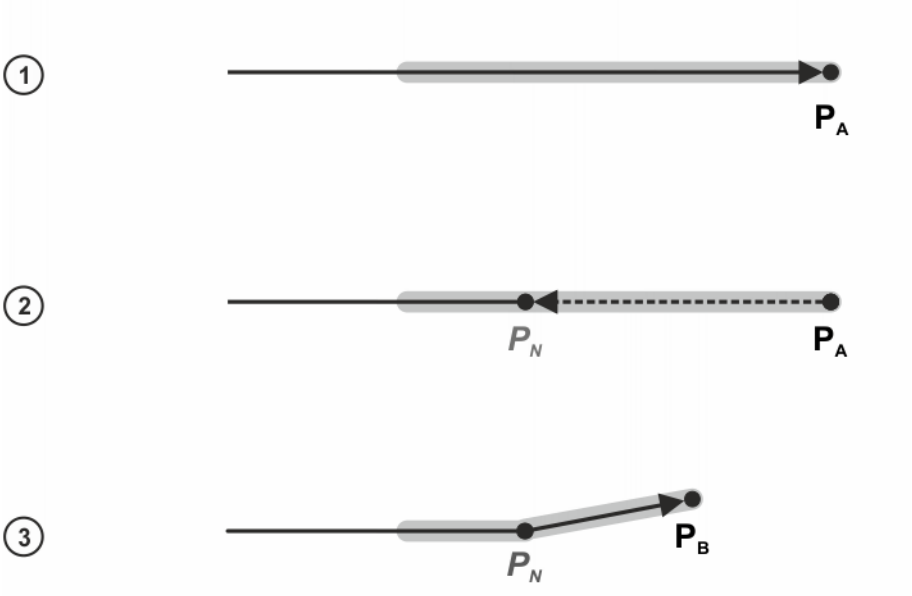


Fig. 4-29: Deletion on Start from buffer

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.

4.17.12.3 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.

4.18 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 E6) 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.

4.19 Bypassing workspace monitoring



These workspaces are part of the System Software and are independent of workspaces in KUKA.SafeOperation or other safety options.

Description

Workspaces can be configured for a robot. A monitoring function is triggered if the robot violates a workspace. Exactly what reactions occur when the monitoring function is triggered depends on the configuration.
(>>> *6.15 "Configuring workspaces" Page 257*)

One possible reaction, for example, is that the robot stops. In this case, the workspace monitoring function must be bypassed 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.

4.20 Display functions

4.20.1 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.

4.20.1.1 Window Actual position, view Cartesian

Information about the current Cartesian position is displayed.

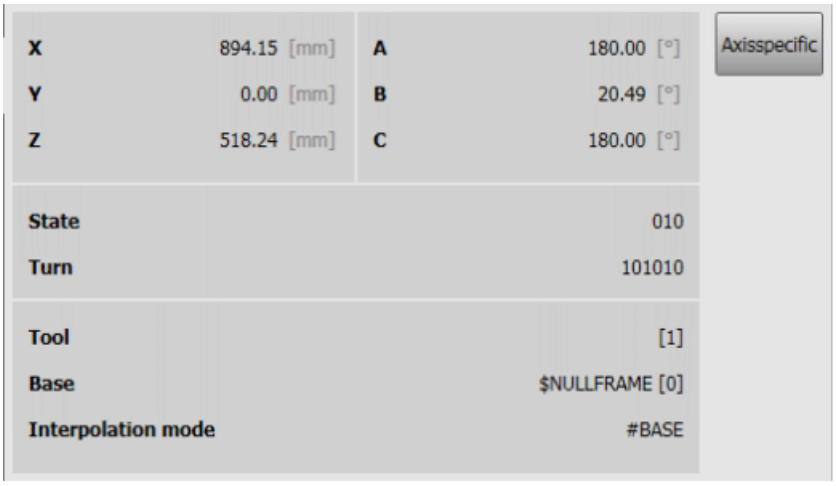


Fig. 4-30: Window Actual position, view Cartesian

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 107*)
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.

4.20.1.2 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.

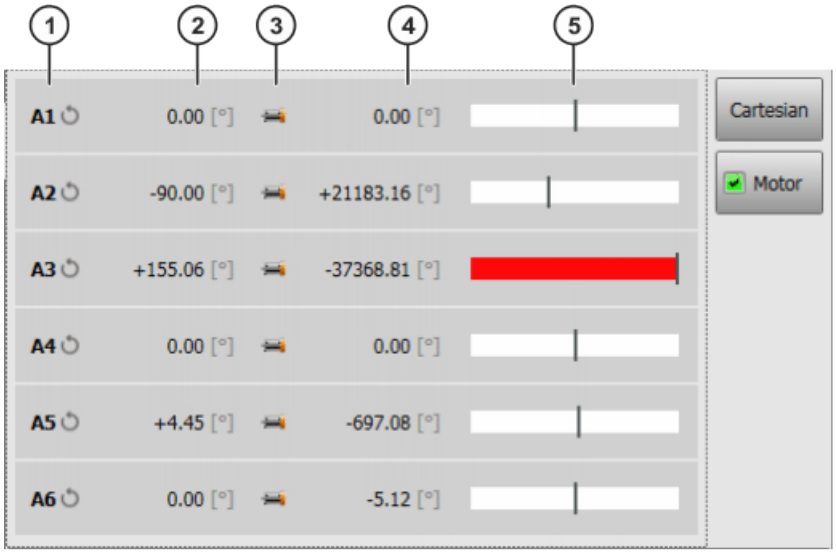


Fig. 4-31: Window Actual position, view Axis-specific

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 108*)

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. 4-32: 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. 4-33: 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.

4.20.2 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

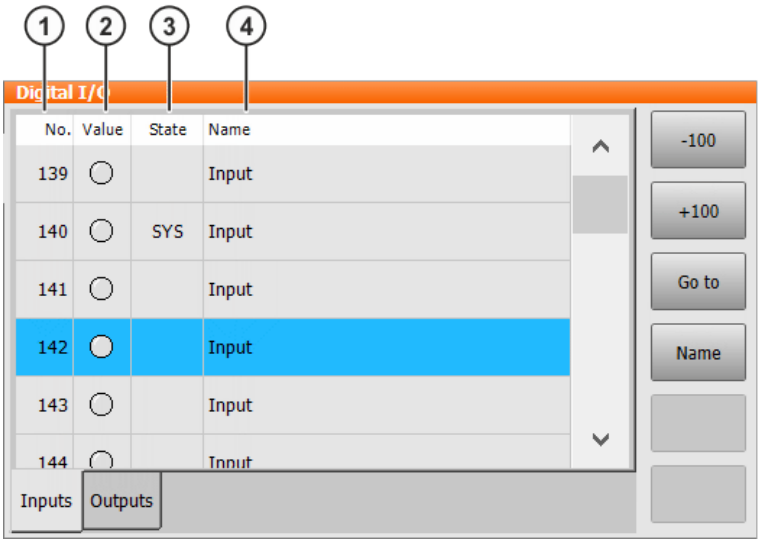


Fig. 4-34: Digital inputs

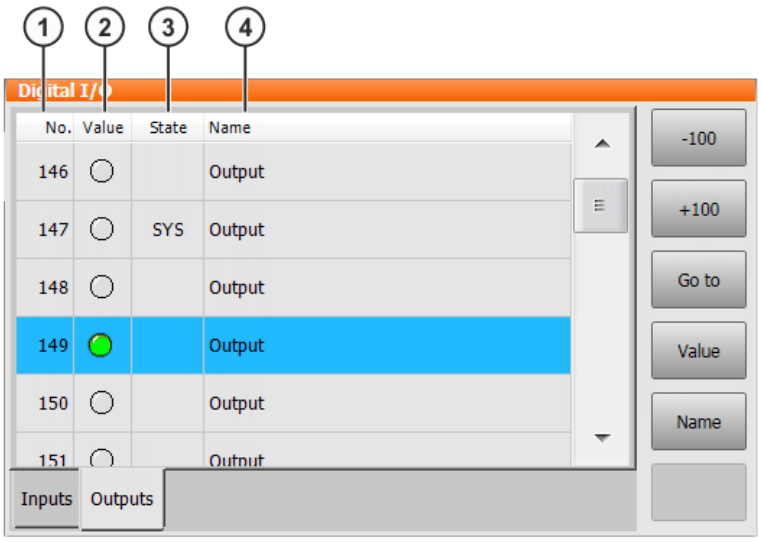


Fig. 4-35: Digital outputs

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	• **SYS**: The value of the input/output is saved in a system variable. This input/output is write-protected. • **RO**: The output is protected against manual modifications. Its value can no longer be modified here in the **Digital I/O** window. The name can still be changed here. The value of the output can still be changed via the variable display and via a program. RO = read only (>>> *6.9 "Protecting a digital output against manual modifications" Page 248*)
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**	

4.20.3 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**	

4.20.4 Displaying inputs/outputs for Automatic External

Procedure

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

Description

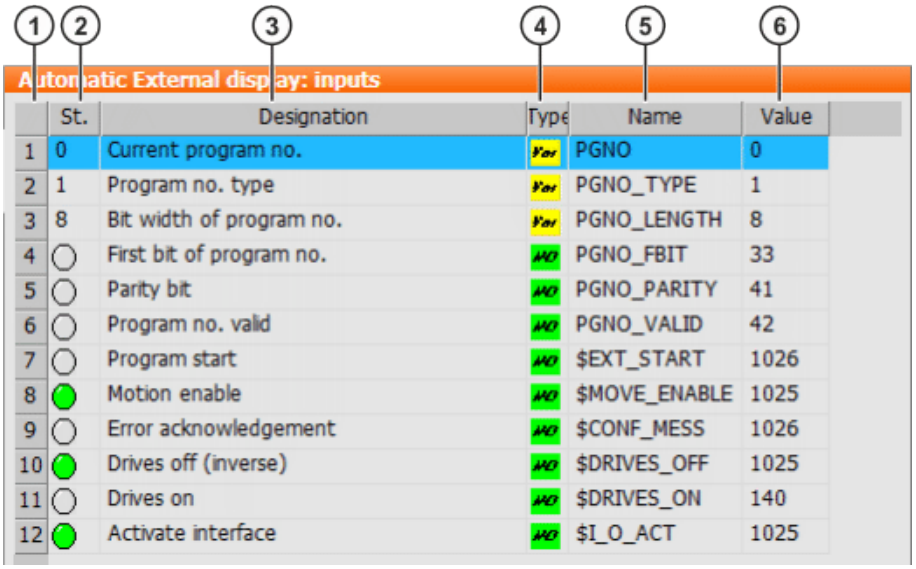


Fig. 4-36: Automatic External inputs (detail view)

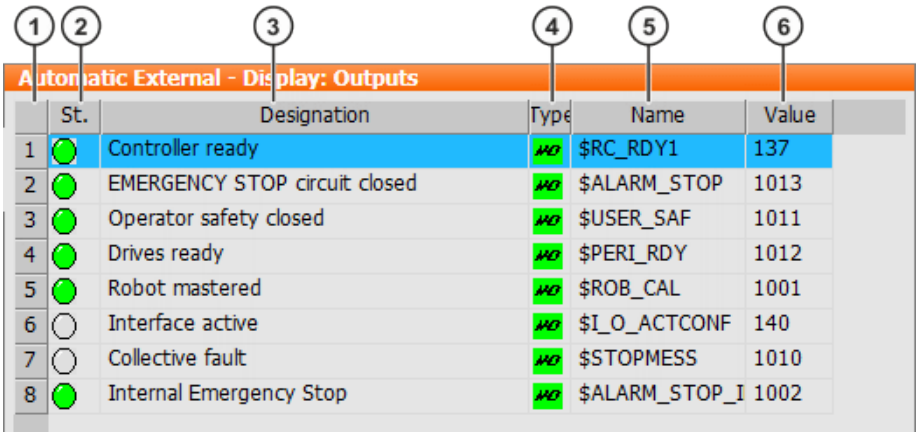


Fig. 4-37: Automatic External outputs (detail view)

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. (>>> *6.22.2 "Configuring Automatic External inputs/outputs" Page 278*)
Inputs/Outputs	Toggles between the windows for inputs and outputs.
Details/Normal	Toggles between the **Details** and **Normal** views.

4.20.5 Displaying and modifying the value of a variable

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

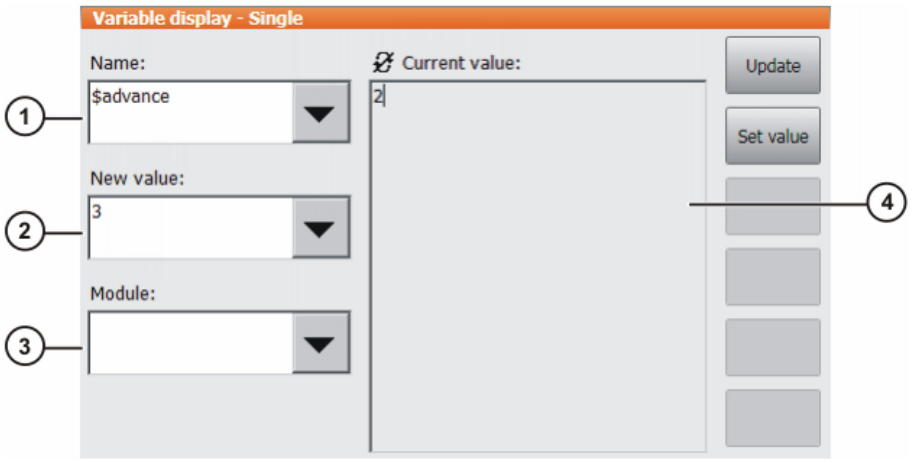


Fig. 4-38: Variable Overview - Single window

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**.

4.20.6 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.

4.20.7 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.

(>>> *6.10 "Configuring the variable overview" Page 249*)

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

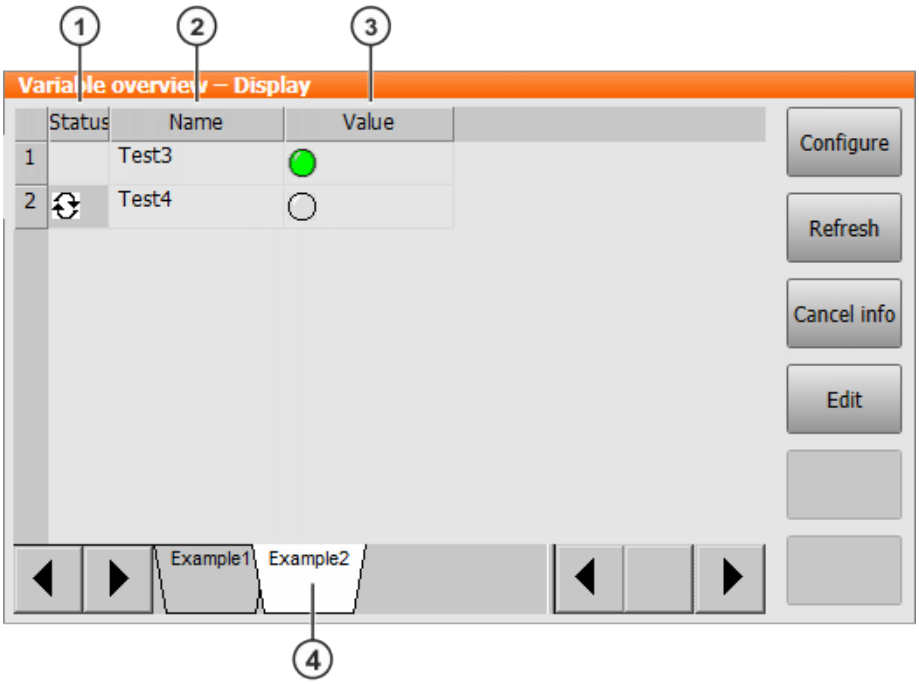


Fig. 4-39: "Variable overview – Display" window

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. (>>> *6.10 "Configuring the variable overview" Page 249*)
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.

4.20.8 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

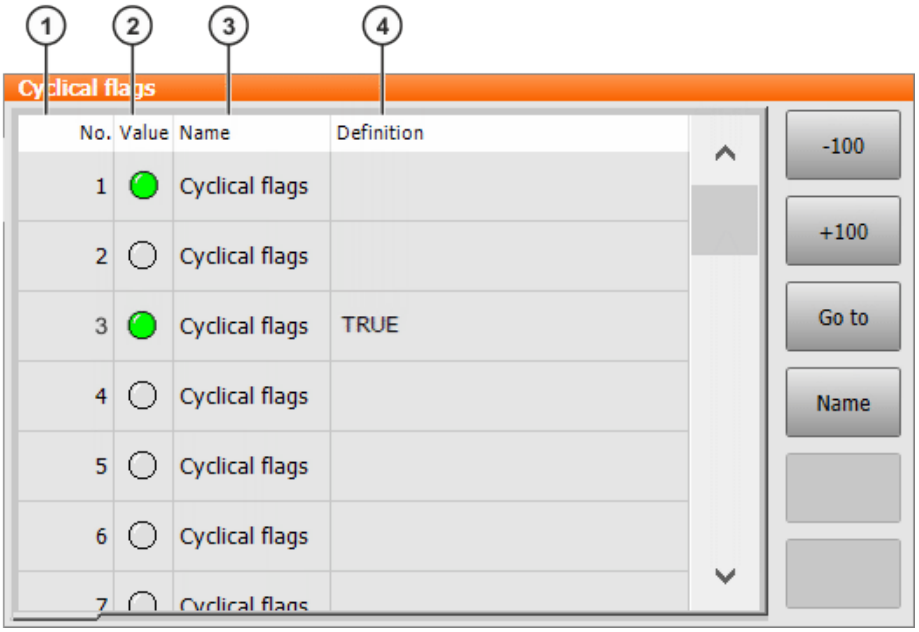


Fig. 4-40: Cyclical flags

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**

4.20.9 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

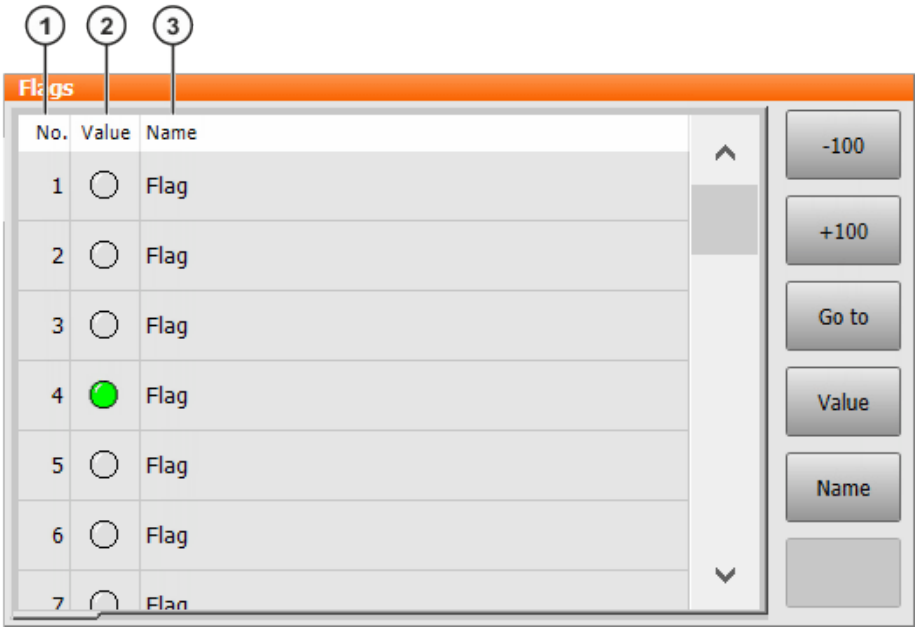


Fig. 4-41: Flags

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**	

4.20.10 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

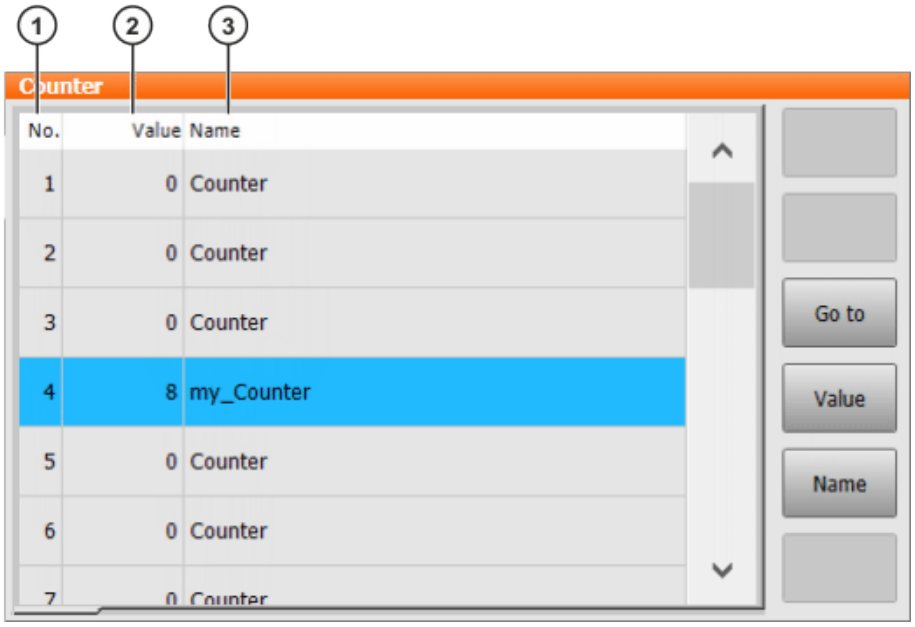


Fig. 4-42: Counter

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**	

4.20.11 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

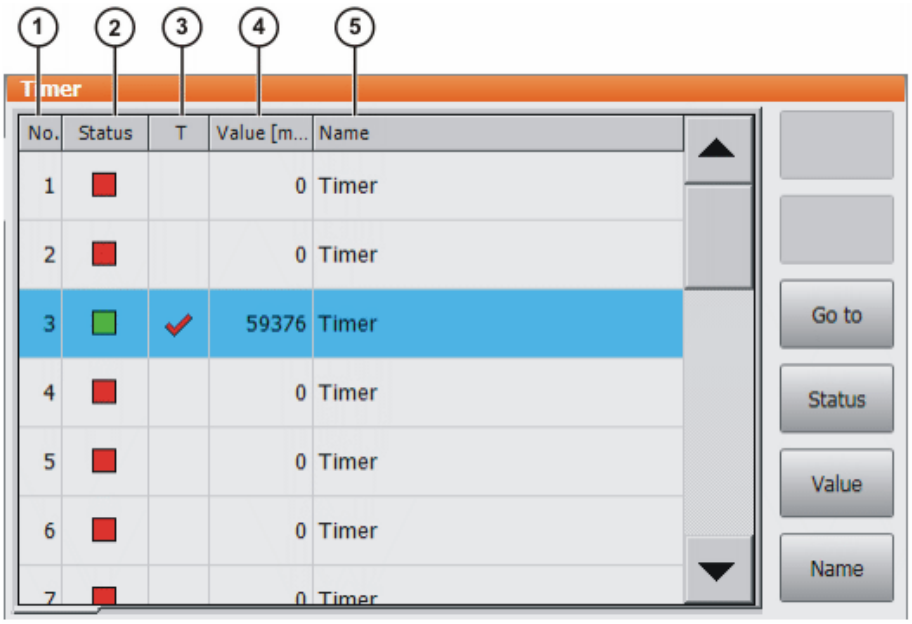


Fig. 4-43: Timer

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**	

4.20.12 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	• Control PC name • Controller name • Operating system version • Version of the safety update (KUKA.SecurityUpdate) if installed • SID • Operating hours • Motherboard • Processor • Graphical representation of the memory capacities: RAM, hard drives, utilization of the kernel system memory
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.
Licenses	The KUKA license conditions and the license conditions of the open-source software used are displayed here.

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

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

4.20.13 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

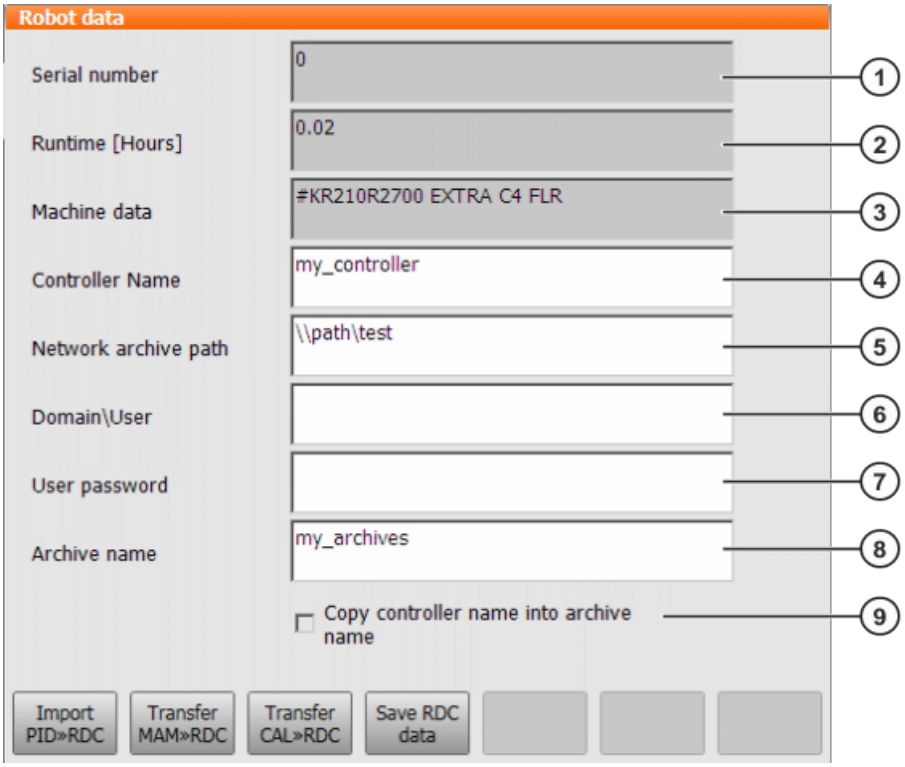


Fig. 4-44: "Robot data" window

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.

4.20.14 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. 4-45*) 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 KUKA.PROFINET.

Robots of an identical type under identical conditions but with different versions of the System Software may have different energy consumption.
The reason for this is that the energy model is adapted to the control hardware typical at the time of software creation. Newer controller generations may have a lower consumption.

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.

(>>> *4.20.14.2 "$ENERGYMODULE: energy model available for robot?" Page 126*)

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 the variable display). 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

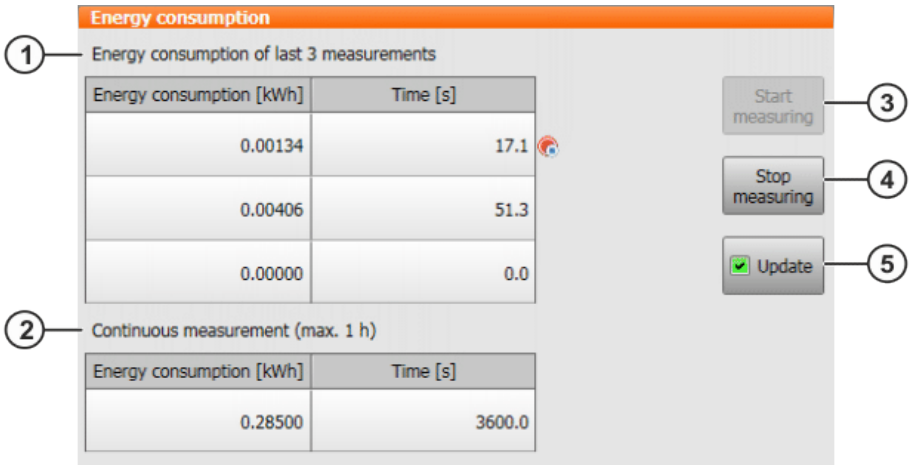


Fig. 4-45: “Energy consumption” window

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.

4.20.14.1 $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)

4.20.14.2 $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 $ENERGY_CONFIG_STATE. (>>> *4.20.14.3 "$ENERGY_CONFIG_STATE: Status of the energy model for each axis " Page 127*)

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

4.20.14.3 $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.

(>>> *4.20.14.2 "$ENERGYMODULE: energy model available for robot?" Page 126*)

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.

4.20.14.4 $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)

4.20.14.5 $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

4.20.14.6 $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.

(>>> *4.20.14.7 "INTERIMENERGY: setting a time stamp for the measurement of energy consumption" Page 130*)

The initial value of $ENERGY_INTERIM 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)
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

4.20.14.7 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.

(>>> *4.20.14.6 "$ENERGY_INTERIM: energy consumption between time stamps" Page 129*)

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.

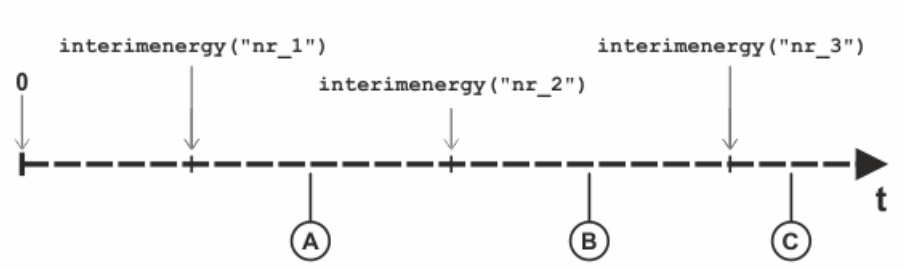


Fig. 4-46: Chronological sequence

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.

4.21 Automatic firmware update for hardware components

Description

If a KUKA hardware component has been added to the system or exchanged, the system software checks whether this hardware has out-of-date firmware. Furthermore, the firmware of the existing hardware is checked if the system software has been updated.

If the system software detects out-of-date firmware, it prepares an update and displays the following message:

An automatic firmware update will be started in {Countdown} seconds.

The counter starts at 60 seconds. While the counter is running, the user has the option of aborting the process and starting it manually at a later time.

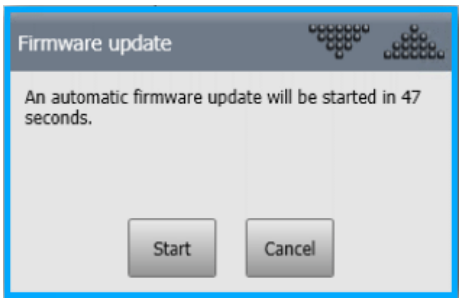


Fig. 4-47: Countdown message

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.
- Following the update, an automatic reboot can be carried out. Whether this is the case depends on the specific situation.

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.

4.21.1 Starting the update immediately

	Do not switch off the robot controller during the update.

Procedure

1. Press **Start** in the window with the Countdown message.
   Or: Wait until the second counter has elapsed. The update then starts automatically.
2. Observe the message window: It displays information about the status of the update.
   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}.
3. Following the update, an automatic reboot can be carried out. Whether this is the case depends on the specific situation.

4.21.2 Starting the update later (manually)

	Do not switch off the robot controller during the update.

Precondition

- User group “Administrator”

Procedure

1. Press **Cancel** in the window with the Countdown message. (This step is still possible for all user groups.)
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 information about the status of the update.

   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}.
5. Following the update, an automatic reboot can be carried out. Whether this is the case depends on the specific situation.

4.21.3 “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.)

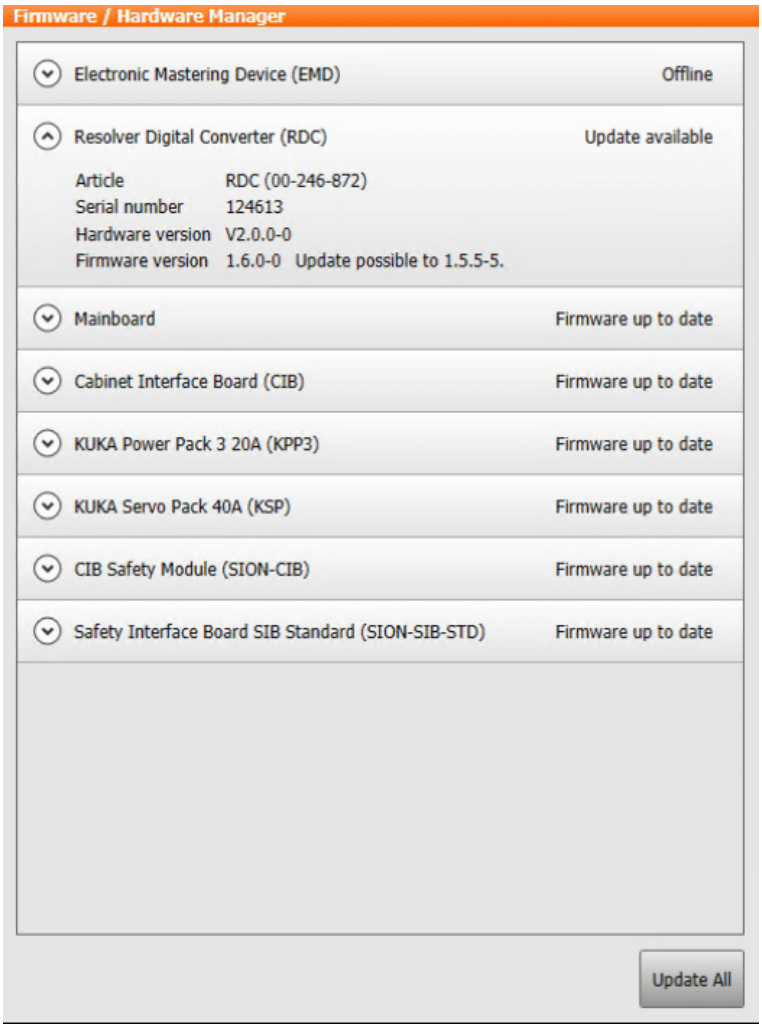


Fig. 4-48: “Firmware / Hardware Manager” window

4.22 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 Safety acceptance overview (without safety option)

Description

The system must not be put into operation until the safety acceptance procedure has been completed successfully. Using checklists, check and document that all safety functions are configured correctly and work correctly.

Safety acceptance must be carried out in the following cases:

- Following initial start-up and recommissioning of the industrial robot
- After a change to the industrial robot
- After a change to the safety configuration
  - The change log must be checked after every modification of the safety configuration.
  - It is also advisable to print out the data set containing the safety parameters using WorkVisual.
- After a software update, e.g. of the System Software

  Safety acceptance after a software update is only necessary if the activation code of the safety configuration changes as a result of the update.

Checklists

The safety acceptance procedure can be carried out and documented using the checklists in the appendix or using your own checklists.

- The items in the checklist must be completed fully and confirmed in writing.
- The completed checklists, confirmed in writing, must be kept as documentary evidence.



The checklists listed below only apply if no safety option is used.
If a safety option, e.g. SafeOperation, is used, other checklists apply. These checklists are contained in the documentation for the relevant safety option.

Checklists in the appendix:

- Checklist for robot and system
  (>>> *15.1.2 "Checklist: Robot and system" Page 729*)
- Checklist for safety functions
  (>>> *15.1.3 "Checklist: Safety functions" Page 730*)
- Checklist for the maximum axis velocities for T1
  (>>> *15.1.4 "Checklist: Maximum axis velocities for T1" Page 732*)

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!

DANGER

Risk of fatal injury due to non-operational safety functions or external safeguards
In the absence of operational safety functions or safeguards, the industrial robot can cause death, severe injuries or damage to property.

- If safety functions or safeguards are dismantled or deactivated, do not operate the industrial robot.

WARNING

Risk of fatal injury due to non-operational external safeguards
Use of Start-up mode disables all external safeguards. Death, severe injuries or damage to property may result.

- There must be no persons in the danger zone of the manipulator while it is in Start-up mode.

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).

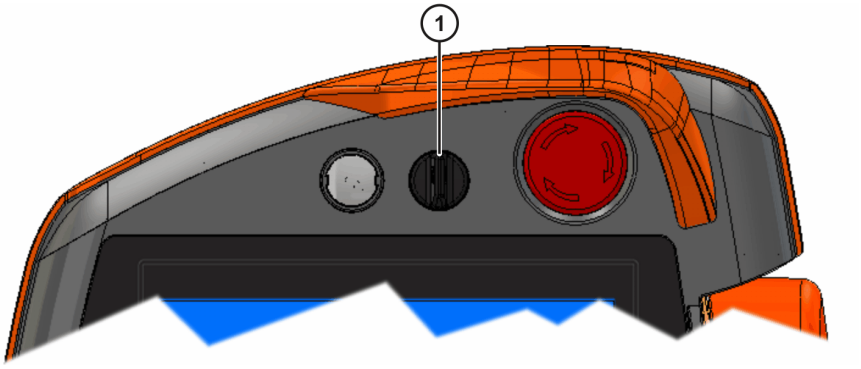


Fig. 5-1: Keyswitch

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.

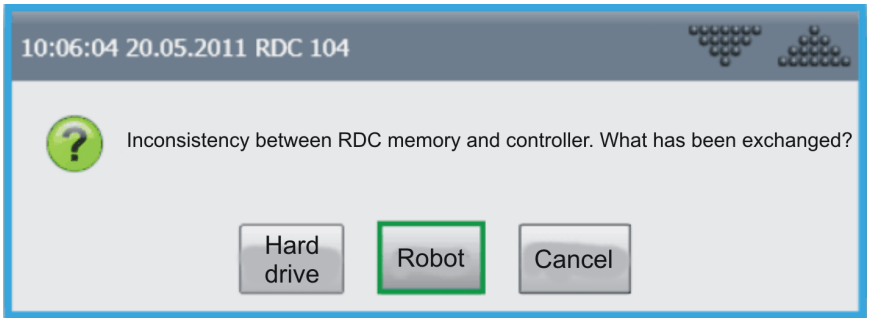


Fig. 5-2: Inconsistency between RDC memory and controller

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*)

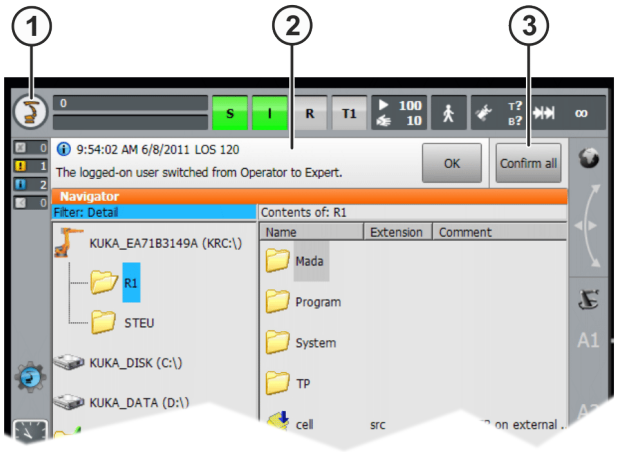


Fig. 5-3: Message window

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.*

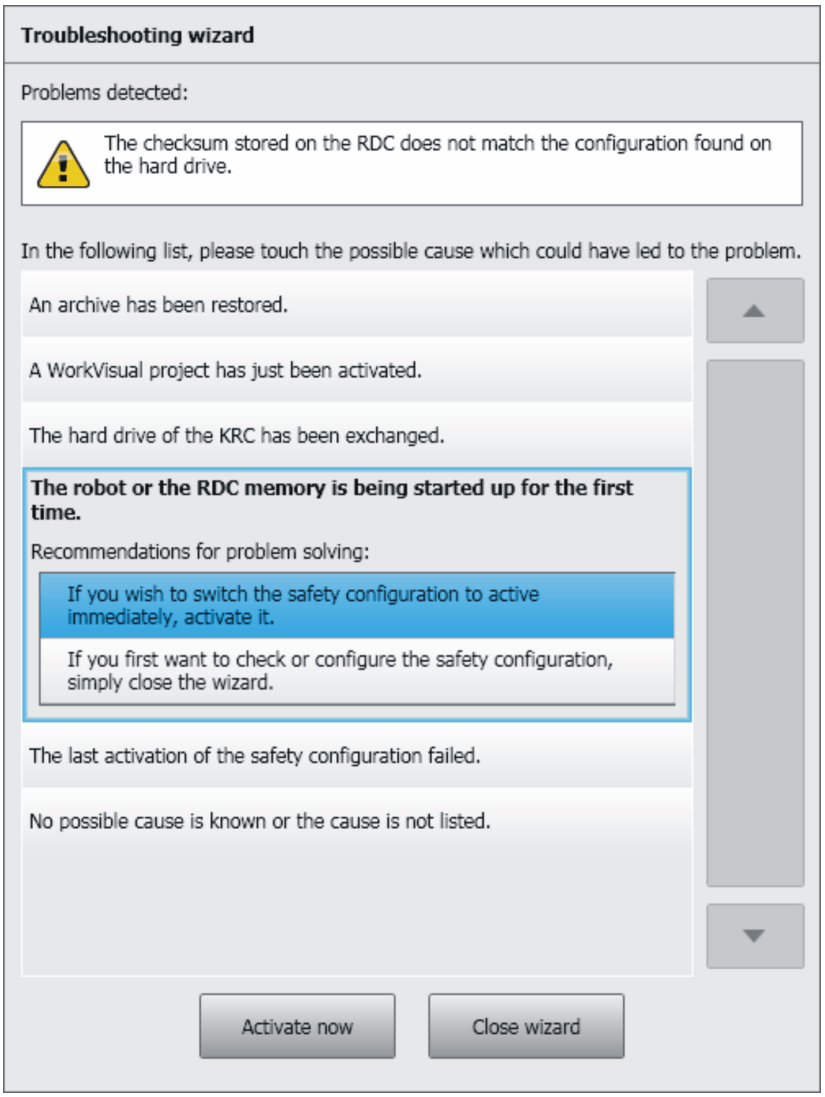


Fig. 5-4: Troubleshooting wizard

- 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*).

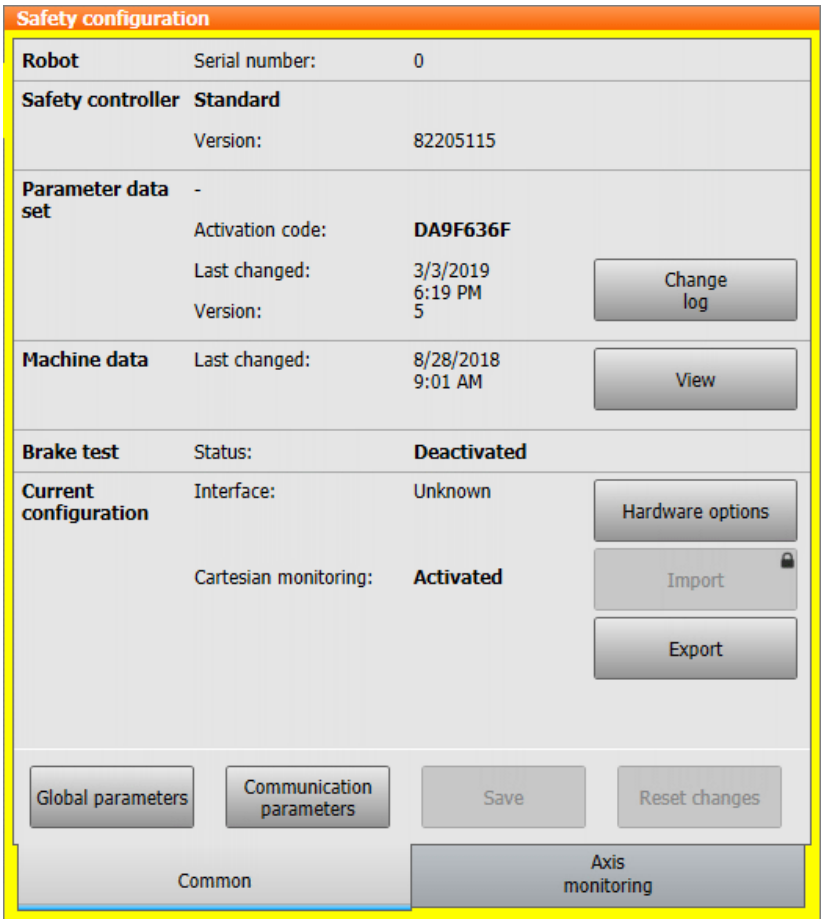


Fig. 5-5: “General” tab

- 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 robot controller is ready for operation. A complete initial start-up can be carried out.

5.3 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 initial passwords: • KukaUser • KukaSystem • target	(>>> *5.4 "Changing the initial password for "KukaUser"" Page 142*) (>>> *5.5 "Changing the initial password for "KukaSystem" and "target"" Page 143*)

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

5.4 Changing the initial password for "KukaUser"

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.

If the initial password is not changed, this allows unauthorized persons to log on to the system.

If the changed password is lost, access to the Windows system is no longer possible – not even for KUKA.

Precondition

- Administrator access rights
- Windows interface
- Mouse and keyboard

Procedure

1. Press the Windows key and the R key simultaneously. The **Run…** window opens.
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 initial password is changed. There is no confirmation message. The change is effective immediately.

(>>> *5.5.1 "Changing the password: return values" Page 144*)

Parameter

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

A password must not contain quotation marks followed by another special character or space. The following examples are thus NOT possible:

- My"&password
- My" password

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.

5.5 Changing the initial password for “KukaSystem” and “target”

Description

The initial passwords of the following users can be changed without (!) having to be entered:

- KukaSystem
- target

Each password can be changed using the procedure described here.

Once an initial password has been changed, further changes are only possible if the current password is known.

If the initial passwords are not changed, this allows them to be changed later by unauthorized persons.

If the changed password is lost, access to the Windows system is no longer possible – not even for KUKA.

Precondition

- Administrator access rights
- Windows interface
- Mouse and keyboard

Procedure for KUKASystem

1. Press the Windows key and the R key simultaneously. The **Run…** window opens.
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="kukasystem" /
op="NEW_PW" /p="NEW_PW" /cp
```

When doing so, enter the desired new password instead of `NEW_PW` in both cases.

4. Confirm by pressing the Enter key.

The initial password is changed. There is no confirmation message. The change is effective immediately.

Procedure for target

1. Press the Windows key and the R key simultaneously. The **Run...** window opens.
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="target" /
op="NEW_PW" /p="NEW_PW" /cp
```

When doing so, enter the desired new password instead of `NEW_PW` in both cases.



For target: the maximum length of the password is 16 characters.

4. Confirm by pressing the Enter key.
5. Reboot the robot controller with the settings **Cold start** and **Reload files**.

(>>> *5.5.1 "Changing the password: return values" Page 144*)

Parameter

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



A password must not contain quotation marks followed by another special character or space. The following examples are thus NOT possible:

- My"&password
- My" password

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.

5.5.1 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.
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.
ERR_TARGETPASSWORDTOO-LONG	0x8000	The password for the user “target” (= VxWorks user) is too long. Maximum length: 16 characters.

*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

Code		Description
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.

5.6 Restricting remote access to the robot controller

Description

In principle, it is possible for external clients to access the robot controller. If, for example, a project is to be transferred from WorkVisual to the robot controller, WorkVisual must access the robot controller.

It is possible to define whether such access is allowed without restriction or only with a login (user name and password).

Default setting:

By default, write access is only possible if the user logs in. Read access is possible without restriction.

Behavior on the external client

If access is only to be granted with a login, the behavior on the external client is as follows:

- When the client initially contacts the robot controller, a login window opens. Access is granted only after a successful login.
- The client saves the login credentials until the end of the session.

Optional: activate the check box next to **Save login credentials** in the login window. The login credentials for this robot controller then also remain saved for future sessions with this client.

The difference between the clocks of the client and the robot controller must not be greater than 5 min. Otherwise access will be denied.
Time zones are automatically converted. Example:

- Host 10:34 MET, Client 09:30 UTC
  Access possible since the difference is 4 min (MET = UTC + 1)
- Host 09:34 MET, Client 09:30 UTC
  Access not possible since the difference is 56 min

Precondition

- User group “Administrator”

Procedure

1. In the main menu, select **Start-up** > **Rights management**.
   The **Rights management** window opens.
2. Make the desired settings on the **Remote access** tab.
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.

The changes are immediately valid.

Remote access tab

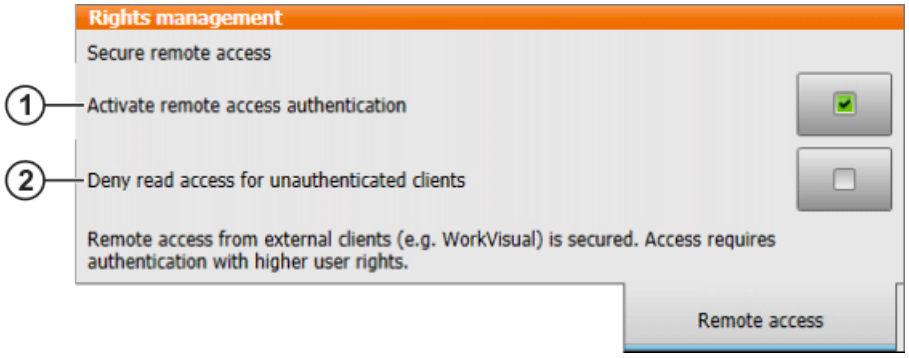


Fig. 5-6: Remote access tab

Item	Description
1	• Check box active: Write access to the robot controller is only allowed for authenticated clients Write access in WorkVisual includes, for example, the transfer of projects from WorkVisual to the robot controller.
2	• Check box active: Read access to the robot controller is only allowed for authenticated clients. Read access in WorkVisual includes, for example, the transfer of projects from the robot controller to WorkVisual or the reading of variables. **Note:** It is always possible to read DeviceInfoService information.

Possible combinations

Blank – blank:

Write access and read access are possible without restriction.

Box	Check box
Activate remote access authentication	
Deny read access for unauthenticated clients	

Check mark - blank (default):

Write access is only possible with login. Read access is possible without restriction.

Box	Check box
Activate remote access authentication	
Deny read access for unauthenticated clients	

Check mark – check mark:

Write access and read access are only possible with login. (= greatest possible restriction)

Box	Check box
Activate remote access authentication	
Deny read access for unauthenticated clients	

Blank – check mark:
This combination is not possible.

“Old” hosts and clients

Access to the following “old” hosts cannot be restricted:

- System Software 8.6 lower than 8.6.5 Build 231
- System Software 8.7 lower than 8.7.0 Build 231

As a rule, the following “old” clients can only access “old” hosts:

- WorkVisual lower than 6.0.5 Build 530

5.7 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.

5.8 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. (>>> *6.18.2 "Configuring warm-up" Page 270*)

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.

5.9 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

Danger to life and limb due to failure to check the safety configuration
After the safety configuration has been modified, it may contain incorrect data in exceptional cases. If it is not checked and, if necessary, corrected, death to persons, severe injuries or considerable damage to property may result.

- Following modifications to the safety configuration, always check the values for the safe axis monitoring functions.

  (>>> *6.5 "Checking the values for the safe axis monitoring functions" Page 244*)

Description

Parameter	Description
Customer interface	Select here which interface is used: • **Automatic** Default interface, in accordance with the robot controller documentation • **Discrete with operating mode output** Output XG11.3 (pins 2, 5, 8, 11): Output of the signal “AUT” (1= manipulator is in AUT or AUT EXT mode) instead of “Periphery enable” • **Discrete C5 micro with JR input** **Note:** Can only be used with a KR C5 micro Input XG58 (pins 1, 3, 5, 7): Connection of the signal “Mastering reference” instead of “External enabling” • **Discrete C5 micro with JR input and operating mode output** **Note:** Can only be used with a KR C5 micro with IFBsafe interface board and connected interface XG11.3. This is a combination of the two previously mentioned options, i.e. with “AUT” output (instead of “Switch on periphery”) and with input “JR” (instead of “External enabling”). • **PROFIsafe with discrete control of second brake** **Note:** Can only be used with a KR C5 in conjunction with PROFIsafe – Output XG11.1 (pins 11, 14, 15, 16): Output of signal “2nd brake A2-A5” instead of “Local EMERGENCY STOP” – Input XG42: Connection of the signal “Safety stop 1” instead of “Reference switch”

Parameter	Description
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. • $CRIT_PERI_ACK_REQ can influence the response of the US2 peripheral contactor. (>>> *5.10.4 "$CRIT_PERI_ACK_REQ" Page 154*) 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 reset 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.

5.10 Preventing an unexpected start of the peripheral devices

5.10.1 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

Danger to life and limb due to missing additional operator action
The start-up of a peripheral device through the simple use of the enabling switch is not permissible if as a result there is a possibility of the device causing damage to property or injury. According to applicable standards, the device may, in this case, only start up after an additional operator action.
If no additional operator action is carried out, property damage, injuries or death to persons may result.

- Introduce an additional operator action, e.g. the “additional enabling” described here.

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.

5.10.2 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: open the $machine.dat file under KRC:\STEU\Mada.

1.
2. Allocate SIGNAL $CRIT_PERI_ACK_REQ (>>> *5.10.4 "$CRIT_PERI_ACK_REQ" Page 154*) either to $IN[1025] or to another input.
   - $IN[1025]: the “additional enabling” is permanently active since $IN[1025] is permanently TRUE
   - Other input: additional enabling can be set to active or inactive via this input
3. Allocate SIGNAL $CRIT_PERI_ACK (>>> *5.10.3 "$CRIT_PERI_ACK" Page 153*) 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.

5.10.3 $CRIT_PERI_ACK

Description

If additional enabling is to be used for starting up peripheral devices, SIGNAL $CRIT_PERI_ACK must be allocated to an output 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.

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]
```

Writability

The system variable is write-protected.

Syntax

`$CRIT_PERI_ACK` == *state*

Explanation of the syntax

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

5.10.4 $CRIT_PERI_ACK_REQ

Description

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

Default in KRC:\STEU\Mada\$machine.dat: in the default setting, the “additional enabling” function is permanently inactive since $IN[1026] is permanently FALSE.

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

Example: SIGNAL $CRIT_PERI_ACK_REQ has been allocated to $IN[66]: if it is desired that the “additional enabling” is permanently active, SIGNAL $CRIT_PERI_ACK_REQ must be allocated to $IN[1025].

```
SIGNAL $CRIT_PERI_ACK_REQ $IN[66]
```

$IN[1025] is permanently TRUE.

Writability

The system variable is write-protected.

Syntax

`$CRIT_PERI_ACK_REQ` == *state*

Explanation of the syntax

Element	Description
state	**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 • **Unknown 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 • **Unknown 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.

5.11 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

Danger to life and limb due to failure to check the safety configuration

After the safety configuration has been modified, it may contain incorrect data in exceptional cases. If it is not checked and, if necessary, corrected, death to persons, severe injuries or considerable damage to property may result.

- Following modifications to the safety configuration, always check the values for the safe axis monitoring functions.

  (>>> *6.5 "Checking the values for the safe axis monitoring functions" Page 244*)

5.12 Switching safe Cartesian monitoring on/off

Description

Safe Cartesian monitoring refers to both KUKA kinematic systems as well as customer kinematic systems (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 on the robot controller 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 a 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

Danger to life and limb due to missing risk assessment
To operate a kinematic system for which safe Cartesian monitoring is not possible, this monitoring can be deactivated. Failure to identify risks that may result from the deactivation may result in death or severe injuries.

- Before deactivating the safe Cartesian monitoring, perform a risk assessment.

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**.

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:**.

5.13 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.

Observe the safety instructions relating to Start-up mode.
External safeguards are disabled in Start-up mode.

- The safety instructions relating to Start-up mode must be observed.
  (>>> *3.12.4.2 "Start-up mode" Page 54*)

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

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 VSS: E2/E22 is closed.

If SafeOperation or SafeRangeMonitoring is used, Start-up mode also influences other signals.

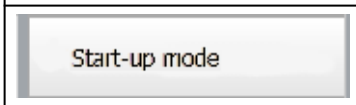
Information about the effects of Start-up mode in conjunction with SafeOperation or SafeRangeMonitoring can be found in the documentation **SafeOperation** and **SafeRangeMonitoring**.

Precondition

- T1 mode
- In the case of VSS: 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.

5.14 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**).

5.15 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 display.
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.
- 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 display.
  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.

5.16 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. 5-7: 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	(>>> *5.16.8 "Reference mastering" Page 177*)
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. (>>> *5.16.10 "Manually unmastering axes" Page 186*)
After an impact with an end stop at more than 250 mm/s	
After a collision.	

5.16.1 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** (>>> *5.16.5 "Mastering with the SEMD" Page 168*)
	Mastering with the dial gauge (>>> *5.16.6 "Mastering with the dial gauge" Page 175*)
	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 (>>> *5.16.9 "Mastering with the MEMD and mark" Page 178*)

SEMD/MEMD

SEMD and/or MEMD are contained in the KUKA mastering kit. There are several variants of the mastering kit.



Fig. 5-8: 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

Damage or interference signals due to incorrect handling of cables

- The following applies to probes to which the signal cable is not permanently attached: The device must always be screwed onto the gauge cartridge without the signal cable attached. Only then may the cable be attached to the device. Otherwise, the signal 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.

NOTICE

Damage to connector due to frequent connection/disconnection

The pluggability of the M8 sensor connector is limited. Frequent connection/disconnection can result in damage to the connector.

- Leave the signal cable connected to the mastering box and disconnect it as little as possible.

NOTICE

Damage due to moving the robot with probe attached

If the following measures are not observed, material damage may result, especially to the probe, signal cable or gauge cartridge.

- If mastering could not be performed successfully, the robot with attached SEMD/MEMD may be moved back to the pre-mastering position for the affected axis.

  The motion may only be carried out manually in T1 mode and with a maximum of 5% jog override.
- For motions other than to the reference notch or back to the pre-mastering position, the SEMD/MEMD must be removed.

5.16.2 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.

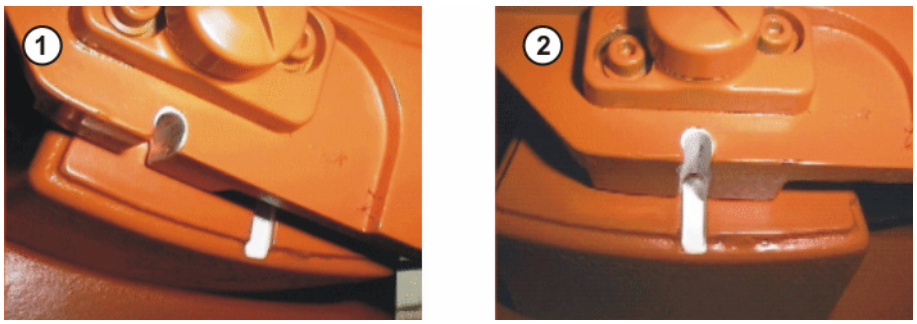


Fig. 5-9: Moving an axis to the pre-mastering position

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.
(>>> *5.16.3 "Moving axes to the pre-mastering position using the probe" Page 166*)

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.

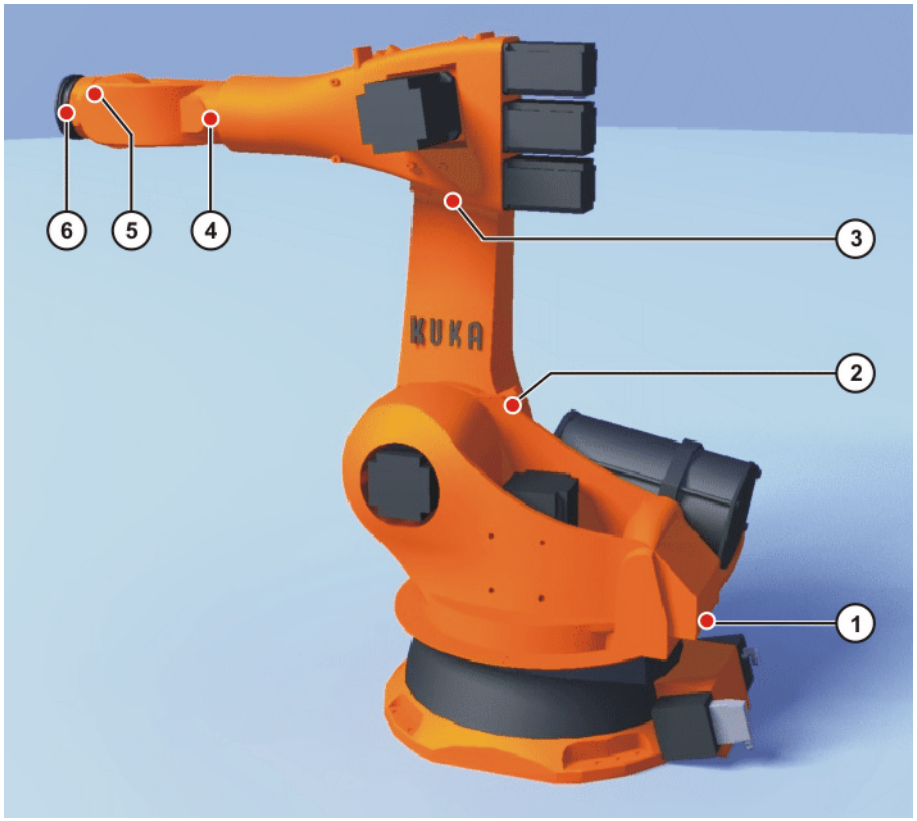


Fig. 5-10: Mastering marks on the robot

Precondition

- User rights: Function group **Jogging with the jog keys**
- T1 mode

NOTICE

Damage to the energy supply system due to rotation
If the energy supply system (if present) is rotated through 360°, it may be torn off or damaged in some other way when A4 and A6 are moved.

- Before A4 and A6 are moved to the pre-mastering position, ensure that the energy supply system is in its correct position and not rotated through 360°.

Procedure

1. Select **Axes** as the reference 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.)

5.16.3 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

Damage to the energy supply system due to rotation
If the energy supply system (if present) is rotated through 360°, it may be torn off or damaged in some other way when A4 and A6 are moved.

- Before A4 and A6 are moved to the pre-mastering position, ensure that the energy supply system 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!
   In other words, 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.

   (>>> *5.16.4 "Mastering LEDs" Page 167*)
5. Jog the robot manually in the minus direction, with a maximum of 5% jog override. 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

Damage to connector due to frequent connection/disconnection
The pluggability of the M8 sensor connector is limited. Frequent connection/disconnection can result in damage to the connector.

- Leave the signal cable connected to the mastering box and disconnect it as little as possible.

5.16.4 Mastering LEDs

For most mastering operations, the smartHMI displays a list of axes. There are 2 LEDs to the right of the list.

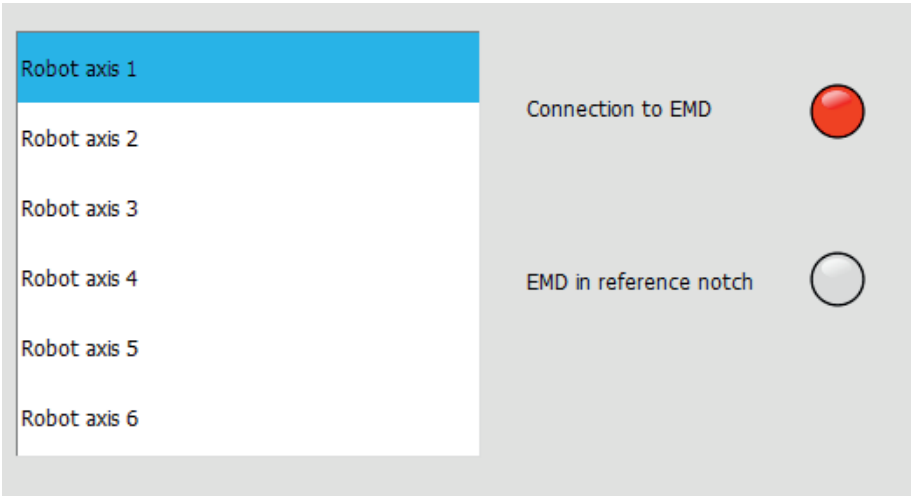


Fig. 5-11: Mastering LEDs

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.

(>>> *5.16.3 "Moving axes to the pre-mastering position using the probe" Page 166*)

5.16.5 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** (>>> *5.16.5.1 "First mastering (with SEMD)" Page 169*) First mastering is carried out without a load.
2	**Teach offset** (>>> *5.16.5.2 "Teach offset (with SEMD)" Page 172*) “Teach offset” is carried out with a load. The difference from the first mastering is saved.
3	If required: **Load mastering with offset** (>>> *5.16.5.3 "Checking load mastering with offset (with SEMD)" Page 173*) “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.

5.16.5.1 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

Damage or interference signals due to incorrect handling of cables

- 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 signal 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 that must be mastered first is selected.

   The order in which the axes must be mastered is defined by the System Software. The sequence is usually in ascending order according to axis numbers. This may differ for a few robot types.

2. Remove the cover from connection X32.

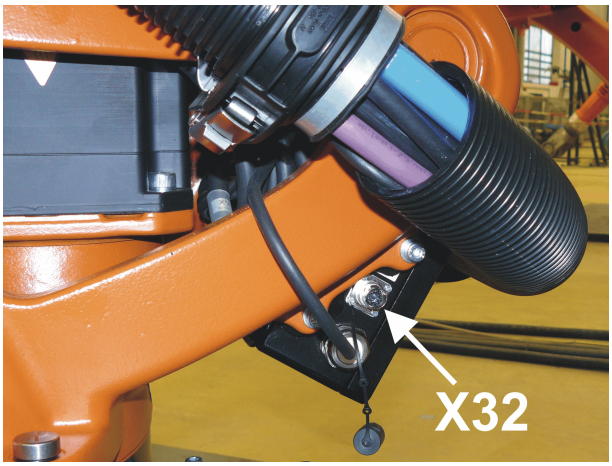


Fig. 5-12: Removing cover from X32

3. Connect the EtherCAT cable to X32 and to the mastering box.

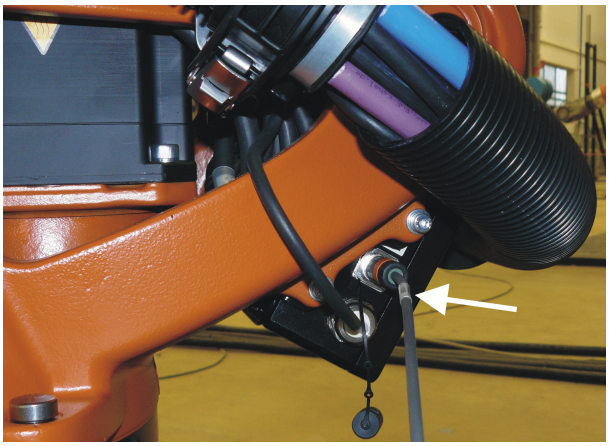


Fig. 5-13: Connecting the EtherCAT cable to X32

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.)

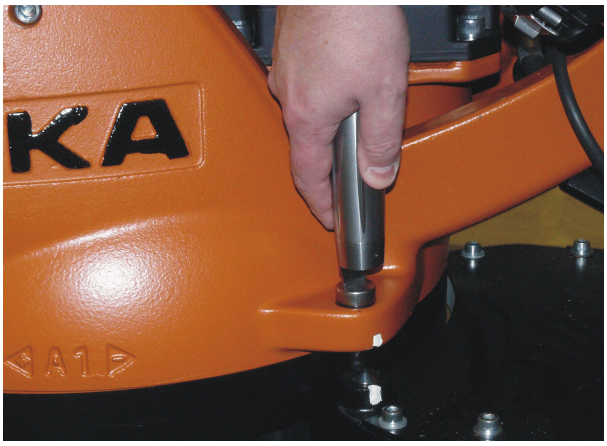


Fig. 5-14: Removing protective cap from gauge cartridge

5. Screw the SEMD onto the gauge cartridge.



Fig. 5-15: 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.

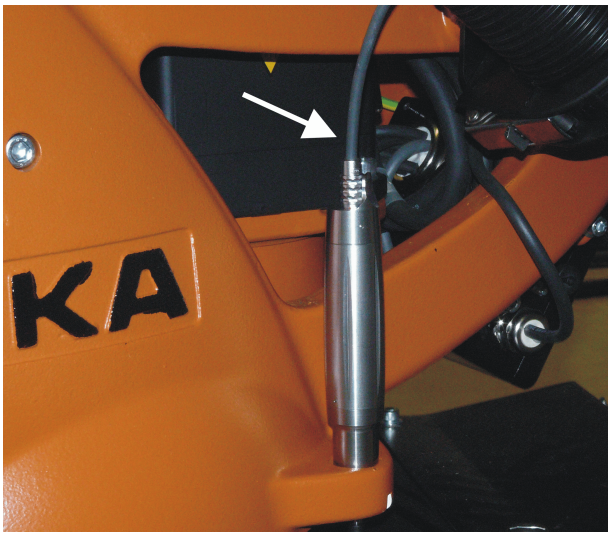


Fig. 5-16: Attaching signal cable to SEMD

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.

   - If mastering could not be carried out successfully, the controller indicates this via a message.

     (>>> *"Mastering not successful" Page 171*)
   - If mastering has been carried out successfully, continue with the following step.
10. The robot controller saves the mastering values and hides the axis in the window.

    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

Damage to connector due to frequent connection/disconnection
The pluggability of the M8 sensor connector is limited. Frequent connection/disconnection can result in damage to the connector.

- Leave the signal cable connected to the mastering box and disconnect it as little as possible.

Mastering not successful

If mastering could not be carried out successfully, the controller indicates this via a message.

- In this case, move the robot back to the pre-mastering position for the affected axis and perform the mastering again.

NOTICE

Damage due to moving the robot with probe attached
If the following measures are not observed, material damage may result, especially to the probe, signal cable or gauge cartridge.

- If mastering could not be performed successfully, the robot with attached SEMD/MEMD may be moved back to the pre-mastering position for the affected axis.

  The motion may only be carried out manually in T1 mode and with a maximum of 5% jog override.
- For motions other than to the reference notch or back to the pre-mastering position, the SEMD/MEMD must be removed.

5.16.5.2 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

Damage or interference signals due to incorrect handling of cables

- 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 signal 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 that must be mastered first is selected.

The order in which the axes must be mastered is defined by the System Software. The sequence is usually in ascending order according to axis numbers. This may differ for a few robot types.

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.

   - If mastering could not be carried out successfully, the controller indicates this via a message.

     (>>> *"Mastering not successful" Page 171*)
   - If mastering has been carried out successfully, continue with the following step.
10. A window opens. The deviation of this axis from the first mastering is indicated in degrees and increments.

    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

Damage to connector due to frequent connection/disconnection
The pluggability of the M8 sensor connector is limited. Frequent connection/disconnection can result in damage to the connector.

- Leave the signal cable connected to the mastering box and disconnect it as little as possible.

5.16.5.3 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.

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

Damage or interference signals due to incorrect handling of cables

- 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 signal 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 to start with is selected.

   The order in which the axes must be mastered is defined by the System Software. The sequence is usually in ascending order according to axis numbers. This may differ for a few robot types.
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.

   - If mastering could not be carried out successfully, the controller indicates this via a message.

     (>>> *"Mastering not successful" Page 171*)
   - If mastering has been carried out successfully, continue with the following step.
10. The difference from “Teach offset” is displayed.

    If required, press **Save** to save the values. The old mastering values are deleted.

    To restore a lost first mastering, always save the values.

Usually axes A4, A5 and A6 are mechanically coupled in the forwards direction. 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.
There may be other couplings for a few robot types.

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

Damage to connector due to frequent connection/disconnection
The pluggability of the M8 sensor connector is limited. Frequent connection/disconnection can result in damage to the connector.

- Leave the signal cable connected to the mastering box and disconnect it as little as possible.

5.16.6 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.

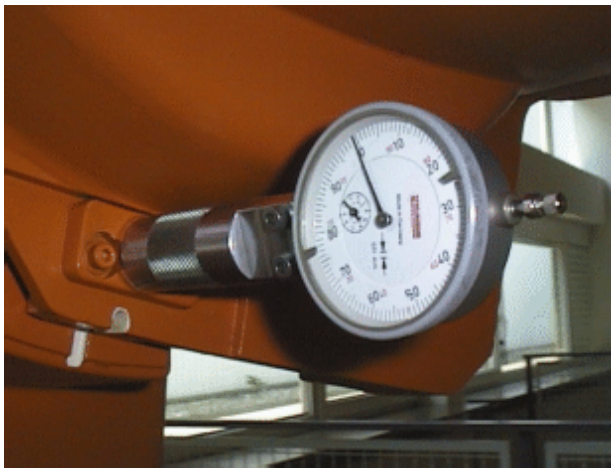


Fig. 5-17: Dial gauge

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 reference 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.

   The order in which the axes must be mastered is defined by the System Software. The sequence is usually in ascending order according to axis numbers. This may differ for a few robot types.

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 zero.
   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.

5.16.7 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.

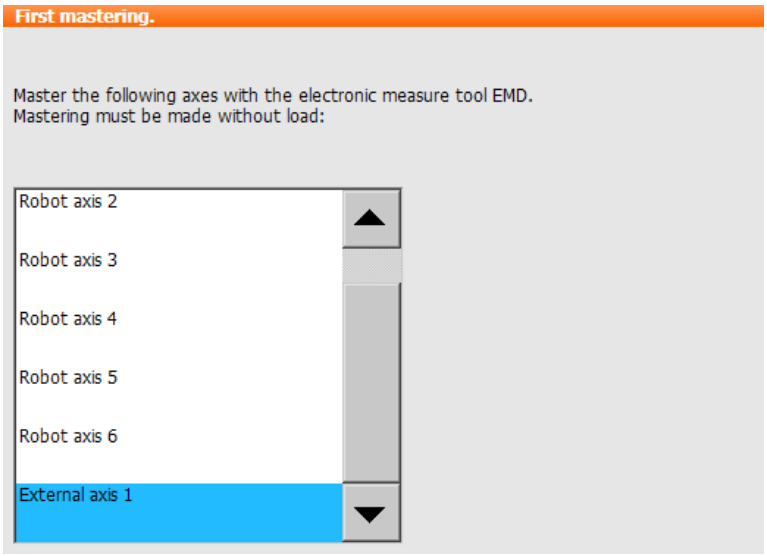


Fig. 5-18: Selection list of axes to be mastered

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.

5.16.8 Reference mastering

The procedure described here must not be used when commissioning the robot.

Description

Reference mastering is suitable if repair work is due on a correctly mastered robot and it is to be expected that the robot will lose its mastering. Example: Exchange of RDC

The robot is moved to the $MAMES position before the repair 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.
  (>>> *4.20.13 "Displaying/editing robot data" Page 121*)

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.

   The order in which the axes must be mastered is defined by the System Software. The sequence is usually in ascending order according to axis numbers. This may differ for a few robot types.

2. Press **Master**. The selected axis is mastered and removed from the option window.
3. Repeat step 2 for all axes to be mastered.

5.16.9 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.
  (>>> *5.16.9.1 "Moving A6 to the mastering position (with line mark)" Page 179*)
- 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** (>>> *5.16.9.2 "First mastering (with MEMD)" Page 180*) First mastering is carried out without a load.
2	**Teach offset** (>>> *5.16.9.3 "Teach offset (with MEMD)" Page 183*) “Teach offset” is carried out with a load. The difference from the first mastering is saved.

Step	Description
3	If required: **Load mastering with offset** (>>> *5.16.9.4 "Checking load mastering with offset (with MEMD)" Page 184*) "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.

5.16.9.1 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.

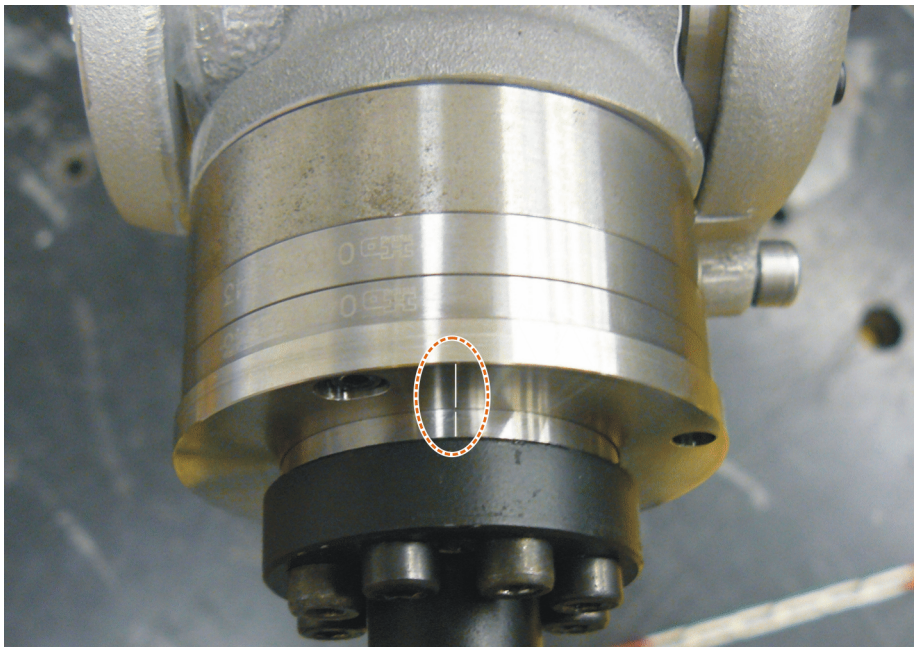


Fig. 5-19: Mastering position A6 – view from above

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.

5.16.9.2 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 that must be mastered first is selected.

   The order in which the axes must be mastered is defined by the System Software. The sequence is usually in ascending order according to axis numbers. This may differ for a few robot types.

2. Remove the cover from connection X32.

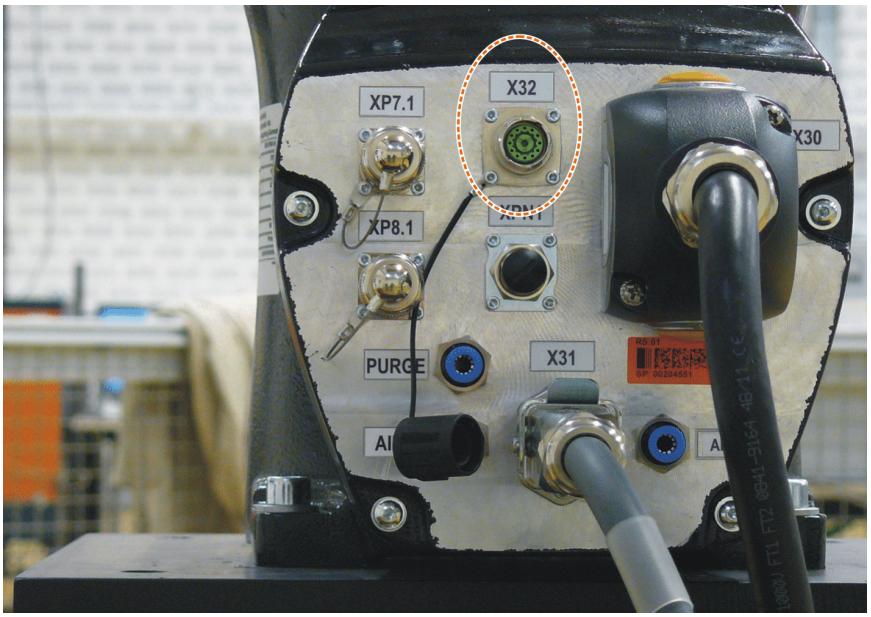


Fig. 5-20: X32 without cover

3. Connect the EtherCAT cable to X32 and to the mastering box.

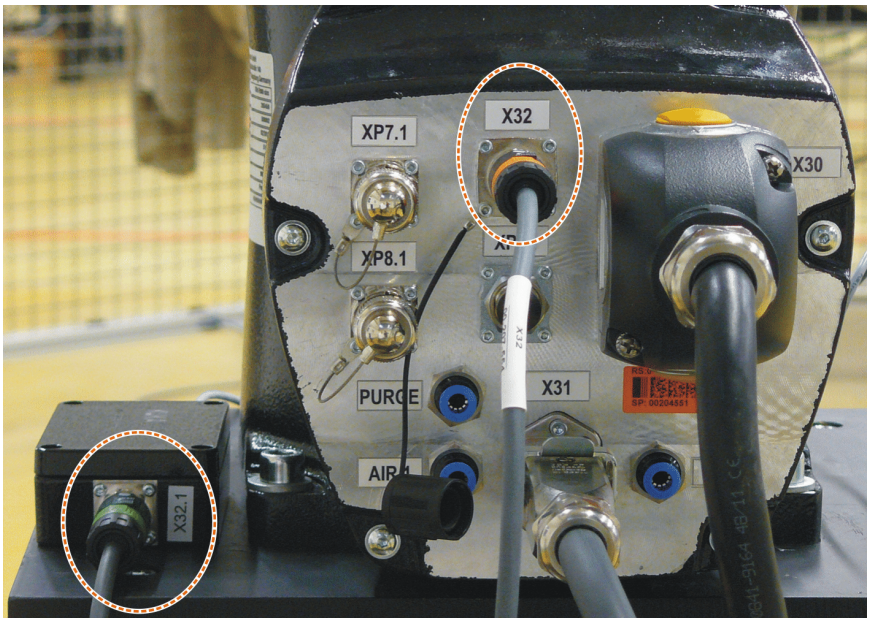


Fig. 5-21: Connecting the cable to X32

4. Remove the protective cap of the gauge cartridge on the axis highlighted in the window.

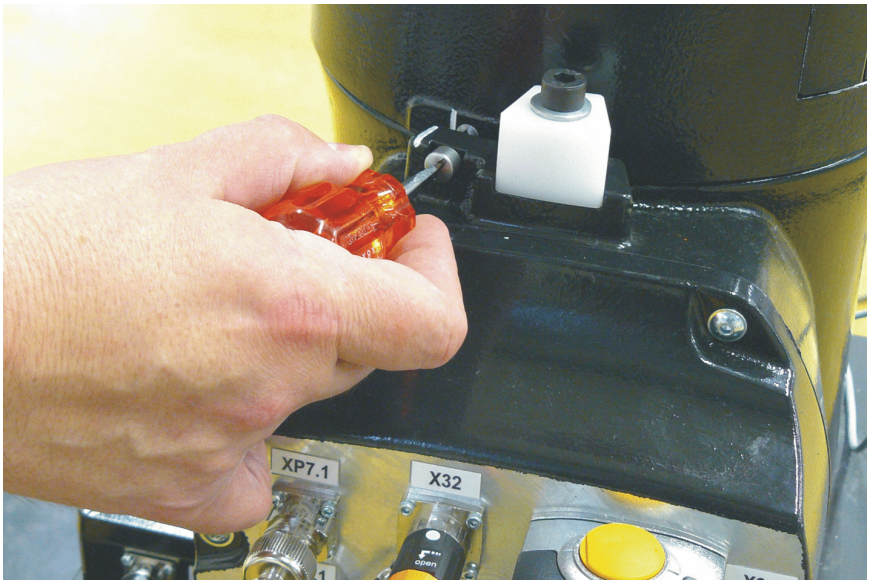


Fig. 5-22: Removing protective cap from gauge cartridge

5. Screw the MEMD onto the gauge cartridge.

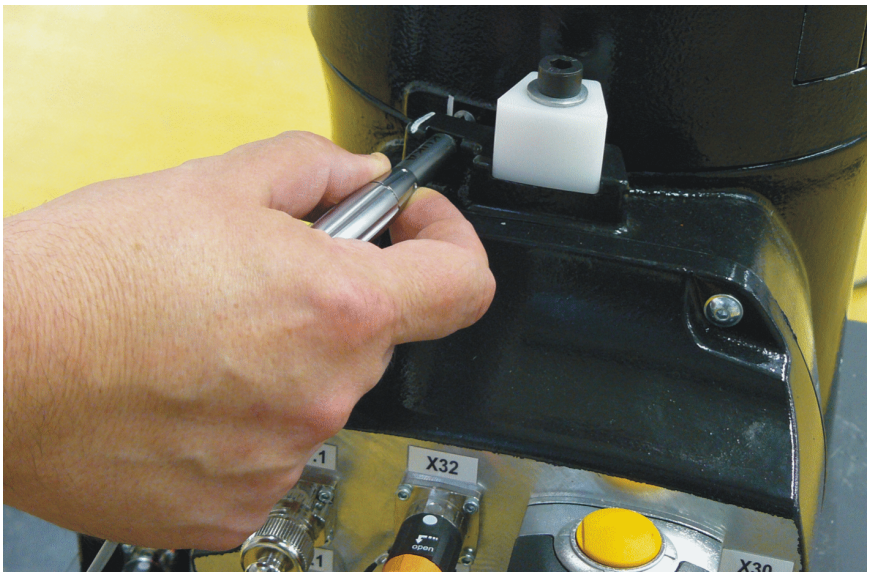


Fig. 5-23: Screwing MEMD onto gauge cartridge

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.

   - If mastering could not be carried out successfully, the controller indicates this via a message.

     (>>> *"Mastering not successful" Page 171*)
   - If mastering has been carried out successfully, continue with the following step.
9. The robot controller saves the mastering values and hides the axis in the window.

   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

Damage to connector due to frequent connection/disconnection

The pluggability of the M8 sensor connector is limited. Frequent connection/disconnection can result in damage to the connector.

- Leave the signal cable connected to the mastering box and disconnect it as little as possible.

5.16.9.3 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 that must be mastered first is selected.

   The order in which the axes must be mastered is defined by the System Software. The sequence is usually in ascending order according to axis numbers. This may differ for a few robot types.
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.

   - If mastering could not be carried out successfully, the controller indicates this via a message.

     (>>> *"Mastering not successful" Page 171*)
   - If mastering has been carried out successfully, continue with the following step.
10. A window opens. The deviation of this axis from the first mastering is indicated in degrees and increments.

    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

Damage to connector due to frequent connection/disconnection
The pluggability of the M8 sensor connector is limited. Frequent connection/disconnection can result in damage to the connector.

- Leave the signal cable connected to the mastering box and disconnect it as little as possible.

5.16.9.4 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.

In the case of robots where A6 has a line mark, the value determined for this axis is not displayed, i.e. the initial 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** > **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 to start with is selected.

The order in which the axes must be mastered is defined by the System Software. The sequence is usually in ascending order according to axis numbers. This may differ for a few robot types.

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.

   - If mastering could not be carried out successfully, the controller indicates this via a message.

     (>>> *"Mastering not successful" Page 171*)
   - If mastering has been carried out successfully, continue with the following step.
10. The difference from “Teach offset” is displayed.

    If required, press **Save** to save the values. The old mastering values are deleted.

    To restore a lost first mastering, always save the values.



Usually axes A4, A5 and A6 are mechanically coupled in the forwards direction. 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.
There may be other couplings for a few robot types.

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

Damage to connector due to frequent connection/disconnection
The pluggability of the M8 sensor connector is limited. Frequent connection/disconnection can result in damage to the connector.

- Leave the signal cable connected to the mastering box and disconnect it as little as possible.

5.16.10 Manually unmastering axes

Description

The mastering values of the individual axes can be deleted. The axes do not move during unmastering.

Usually axes A4, A5 and A6 are mechanically coupled in the forwards direction. 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.
There may be other couplings for a few robot types.

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.

5.17 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. 5-24: Before automatic determination

Item	Description
1	Current negative limit switch
2	Current position of the axis
3	Current positive limit switch

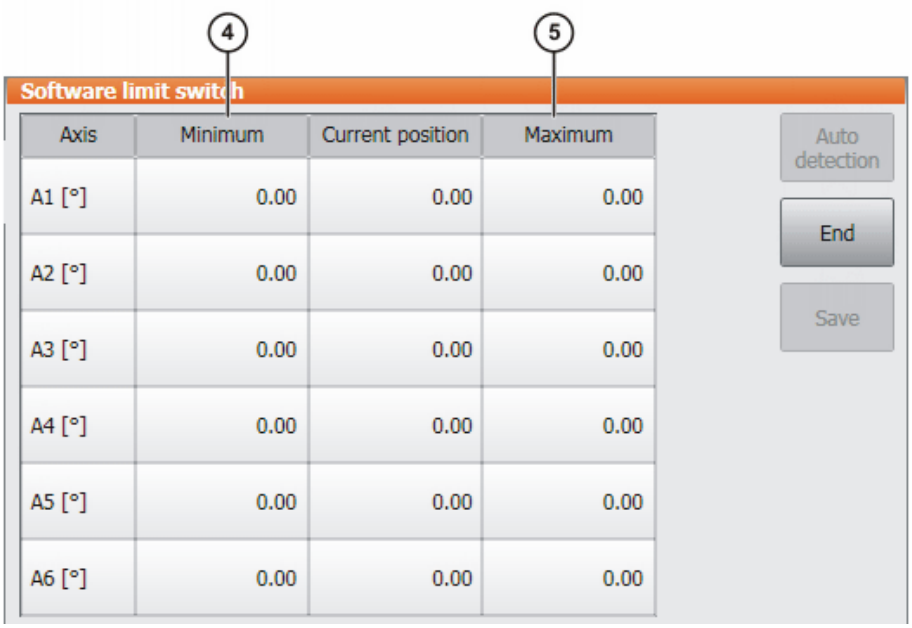


Fig. 5-25: During automatic determination

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.

5.18 Calibration

5.18.1 Defining the tool direction

Information about the tool direction of certain 4-axis robots
For most robots, **+Z_{FLANGE}** is the axis of the FLANGE coordinate system that projects out of the flange.
With some types of 4-axis robots, it is **+X_{FLANGE}** that projects from the flange. For these robots, the following applies:

- It is not helpful to define a tool direction other than X (= default setting).

In many cases, the axis that projects from the flange can be noted from the documentation of the robot: See the information there about the load center of gravity. Otherwise, contact KUKA.

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

Danger to life and limb due to subsequent change of the tool direction
If the tool direction is changed after calibration or after program creation, unexpected changes to the motion characteristics of the robot may occur. Death, severe injuries or damage to property may result.

- Define the tool direction before calibration and before program creation.

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 via the variable display or by writing to the variable from the program.

5.18.2 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

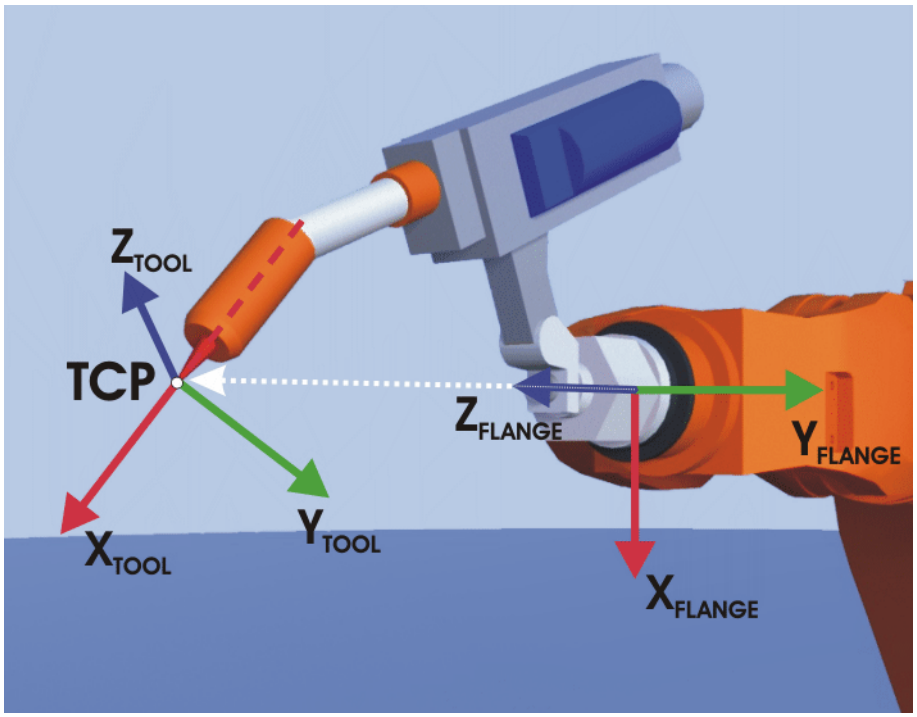


Fig. 5-26: TCP calibration principle

5.18.3 Calibrate TOOL or enter it numerically (tool/workpiece on flange)

Precondition

- User rights: Function group **Calibration**
- Operating mode T1
- Other preconditions dependent on the calibration method: See the descriptions of the individual methods after the overview
  (>>> *5.18.9 "Overview of calibration methods" Page 200*)
  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.
   (>>> *5.18.8 ""Tool/base management" window" Page 195*)
2. Select the **Tool Workpiece** tab.
3. Press the **Add** button. The **Edit tool** window opens.
4. Assign a number and a name for the tool (or workpiece).
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 all values numerically.
   - Or touch the **Calibrate** button, select the method and perform calibration.
     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.

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.

7. Save.
   Information about this calibration is now displayed in the **Last measurement** area.

5.18.4 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].

5.18.5 Calibrate BASE or enter it numerically (base/fixed tool)

Precondition

- User rights: function group **Calibration**
- Operating mode T1
- Other preconditions dependent on the calibration method
  (>>> *5.18.9 "Overview of calibration methods" Page 200*)
  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.
   (>>> *5.18.8 "“Tool/base management” window" Page 195*)
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. Under **Assignment**, select:
   - Either **WORLD**
   - Or, in the case of the BASE for an external kinematic system: the external kinematic system
7. 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.

8. Save.

   Information about this calibration is now displayed in the **Last measurement** area.

5.18.6 Introduction to calibration of external kinematic systems

Description

Calibration of the external kinematic system is necessary to enable the motion of the axes of the kinematic system to be synchronized and mathematically coupled with the robot axes. An external kinematic system can be a turn-tilt table or positioner, for example.



For linear units, the type of calibration described here must not be used. A separate type of calibration must be used for linear units.
(>>> *5.18.10 "Linear unit" Page 210*)

Precondition

- The external kinematic system has been added in WorkVisual and the project has been transferred to the robot controller.

Overview

Calibration of an external kinematic system consists of 2 steps:

Step	Description
1	Calibration of the root point of the external kinematic system (>>> *5.18.7 "Calibrating or numerically entering the root point of an external kinematic system" Page 194*)
2	BASE calibration (>>> *5.18.5 "Calibrate BASE or enter it numerically (base/fixed tool)" Page 192*) If a BASE has been created in WorkVisual for the external kinematic system, it is already present on the robot controller. It can subsequently be calibrated on the robot controller. If no BASE has been created in WorkVisual, this must be carried out on the robot controller. A BASE is absolutely necessary. Important settings: • If there is a workpiece on the external kinematic system: For BASE calibration, select **Base**. • Or, if there is a tool mounted on the external kinematic system, select **Fixed tool**. • In any case, select the external kinematic system under **Assignment**.

5.18.7 Calibrating or numerically entering the root point of an external kinematic system

- User rights: function group **Calibration**
- T1 mode
- The external kinematic system has been added in WorkVisual and the project has been transferred to the robot controller.
- A previously calibrated tool is mounted on the flange.
  Or: the values X, Y, Z, A, B, C of the root point relative to WORLD are known.

(>>> *5.18.9.9 "Method Root point" Page 209*)

Procedure

1. In the main menu, select **Start-up** > **Tool/base management**.
   The **Tool/base management** window opens.
   (>>> *5.18.8 ""Tool/base management" window" Page 195*)
2. Select the **Ext. kinematic systems** tab.
   The external kinematic system is displayed here. Open the list entry.
3. Press the **Edit** button. The **Edit kinematic system** window opens.
4. Change the name if required.
5. In the **Transformation** area, define the values for the origin (X, Y, Z) and the orientation (A, B, C) of the root point:
   - Either enter the values in the boxes.
   - Or touch the **Calibrate** button 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.

6. Save.
   Information about this calibration is now displayed in the **Last measurement** area.

5.18.8 "Tool/base management" window

5.18.8.1 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.

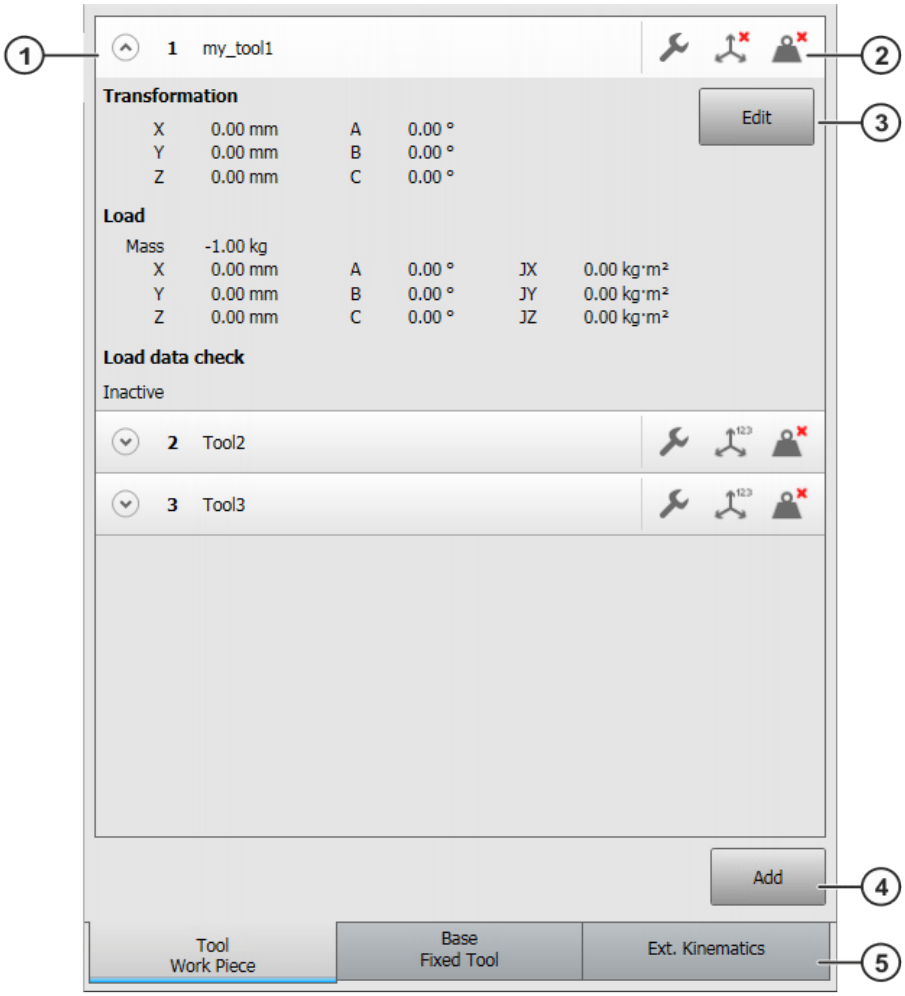


Fig. 5-27: "Overview" area

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. (>>> *5.18.8.2 "Icons in the "Overview" area" Page 196*)

Item	Description
3	Opens the “Edit” view for this tool/workpiece. The data can be edited. (>>> *5.18.8.3 "Tool/base management window – “Editing” area" Page 197*) 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.

5.18.8.2 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 by the user.

Icon	Load data
	The **Mass** box contains the value “-1”. The correct load data must still be entered.

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 If the kinematic system belongs to a RoboTeam, the number to the right indicates the RoboTeam index.
	Robot as RoboTeam participant. A number to the right indicates the RoboTeam index.
	Conveyor

5.18.8.3 Tool/base management window – “Editing” area

The “Editing” area is explained here using the **Tool/Workpiece** tab by way of example.

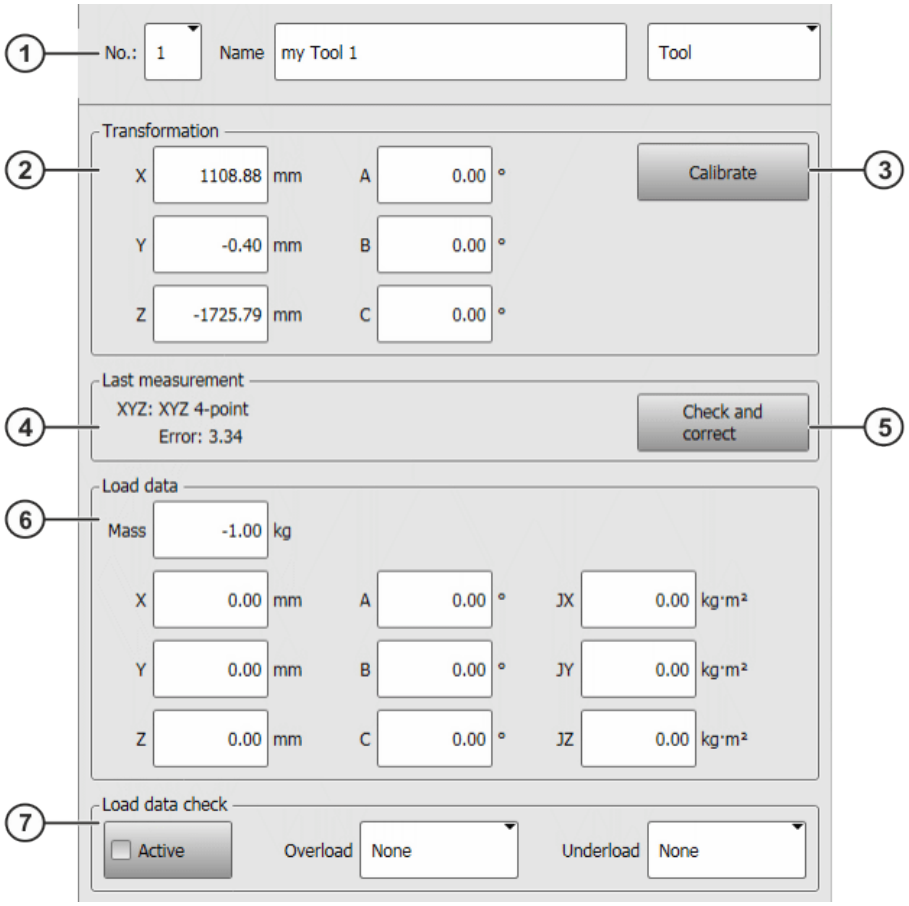


Fig. 5-28: “Editing” area

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. (>>> *5.18.8.4 "Tool/base management window – “Calibration” area" Page 198*)
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.
6	**Tool Workpiece** tab: payload data (>>> *5.19.2 "Entering payload data numerically" Page 215*) **Base Fixed Tool** tab: assignment of BASE to the WORLD coordinate system or an external kinematic system
7	(>>> *5.19.5 "Load data verification" Page 219*)

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.

5.18.8.4 Tool/base management window – “Calibration” area

The “Calibration” area is explained here using the **XYZ 4-point** method by way of example.

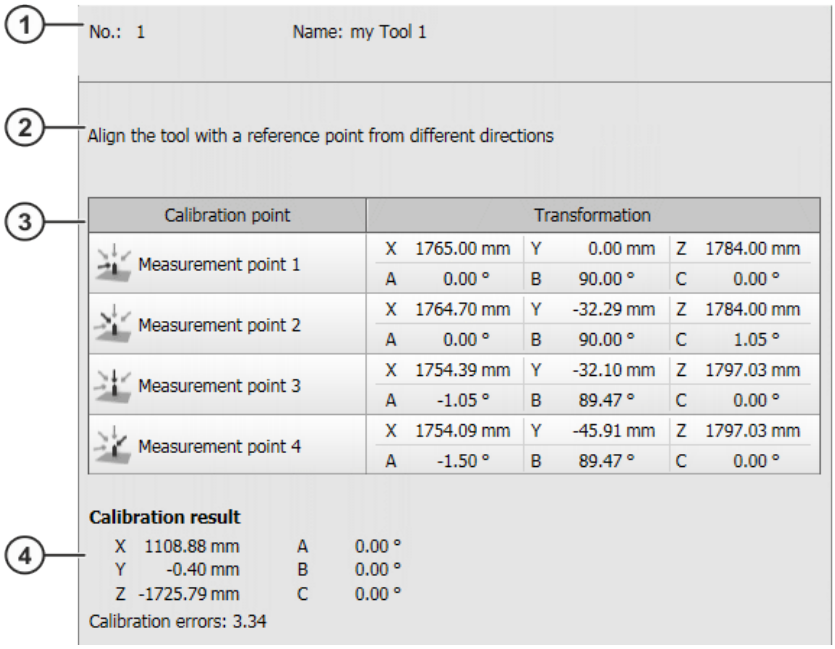


Fig. 5-29: “Calibration” view

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 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 201*)

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 *Measured point {0} reached.* is displayed.

5.18.9 Overview of calibration methods

The table shows which method can be used for which frame and what is calibrated using it.

Method ↓	Tool on flange	Base	Fixed tool	Workpiece on flange
XYZ 4-point	X, Y, Z	- - -	- - -	- - -
XYZ 3-point [1]	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 [2]	- - -	X, Y, Z A, B, C	- - -	X, Y, Z A, B, C
Root point	Only for external kinematic systems. X, Y, Z, A, B, C of the root point are calibrated relative to WORLD.			

[1] The **XYZ 3-point** method is only available if a corresponding kinematic system is used. **XYZ 4-point** is then not available.

[2] The **Indirect** method is not available if the base is assigned to an external kinematic system.

Information about the orientation of certain 4-axis robots
For most robots, $+Z_{FLANGE}$ is the axis of the FLANGE coordinate system that projects out of the flange.
With some types of 4-axis robots, it is $+X_{FLANGE}$ that projects from the flange. For these robots, the following applies:

- The methods for orientation calibration (A, B, C) cannot be applied meaningfully to them in most cases.
- **Recommendation:** Enter the C value numerically. A and B must be 0.

In many cases, the axis that projects from the flange can be noted from the documentation of the robot: See the information there about the load center of gravity. Otherwise, contact KUKA.

5.18.9.1 XYZ 4-point method

Use

- Calibration of X, Y and Z of a tool

The **XYZ 4-point** method is not available for some kinematic systems, in particular for most palletizers. If such a kinematic system is used, the controller automatically provides the **XYZ 3-point** method instead.

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.

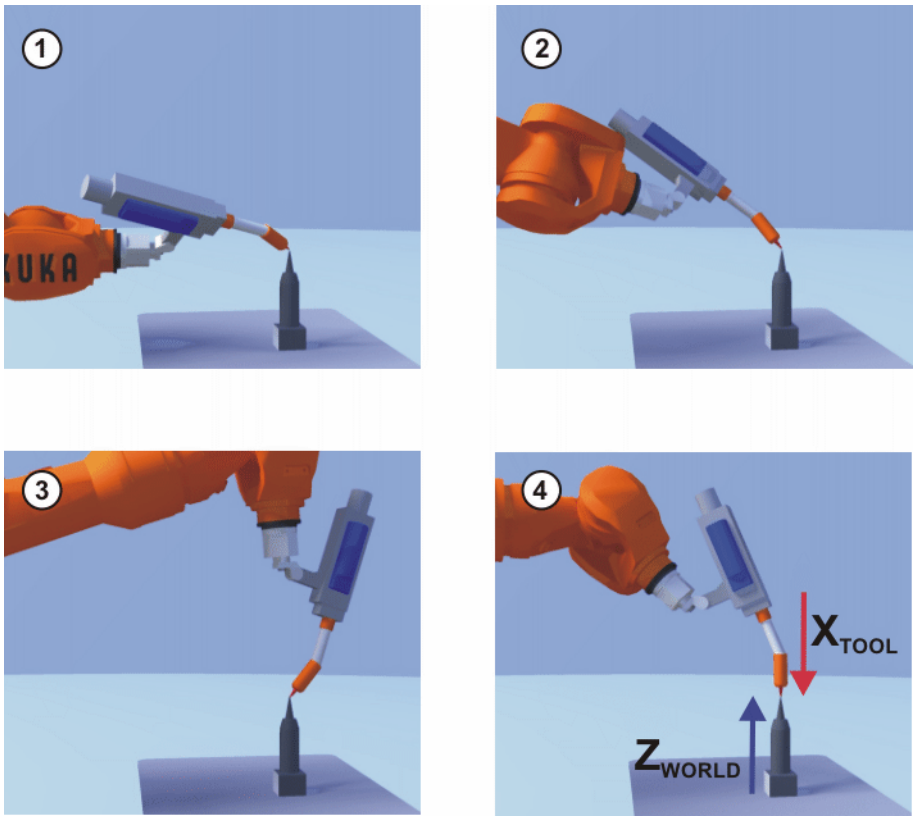


Fig. 5-30: XYZ 4-point method

Measurement result

In the case of the **XYZ 4-point** method, a potential measurement error is displayed in the **Tool/base management** window in the “Calibration” area. (>>> *5.18.8.4 "Tool/base management window – “Calibration” area" Page 198*)

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: Imprecisions earlier in the chain persist further down.
- Mastering: Incorrect mastering leads to imprecise measurement results.
- Calibrating person/“human factor”

5.18.9.2 XYZ 3-point method

Use

Calibration of a tool for the following types of kinematic system:

- 4-axis palletizer
- 5-axis palletizer with palletizing mode active
- In the future, other kinematic systems, in particular most 4-axis kinematic systems

If a corresponding kinematic system is used, the controller provides the **XYZ 3-point** method. **XYZ 4-point** is then not available.

Precondition

- The distance from the TCP to the flange plane is known.
- The tool to be calibrated is mounted on the mounting flange.

Description

- The distance from the TCP to the flange must be entered numerically once the method has been selected.
  Depending on the type of kinematic system, it is the X or Z value. This is not relevant for the entry, however; the important thing is the distance.
- The TCP of the tool to be calibrated is then moved to a reference point with 3 different orientations. The reference point can be freely selected. The distance between the orientations must be at least 15 degrees in each case as standard.

  The controller accordingly calculates the remaining values for the TCP. Depending on the type of kinematic system, these are X and Y or Y and Z.

5.18.9.3 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.

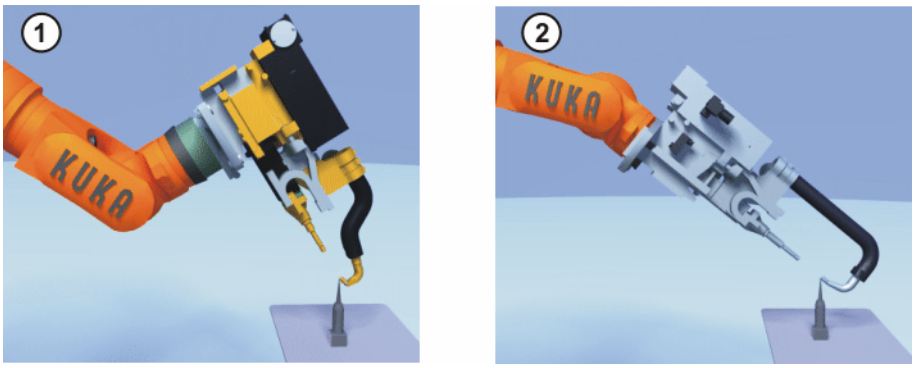


Fig. 5-31: XYZ Reference method

5.18.9.4 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.

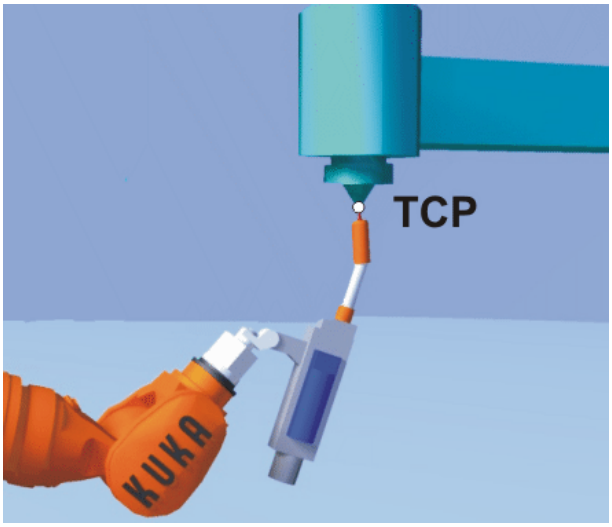


Fig. 5-32: XYZ method

5.18.9.5 ABC world method

Information about the orientation of certain 4-axis robots
For most robots, $+Z_{FLANGE}$ is the axis of the FLANGE coordinate system that projects out of the flange.
With some types of 4-axis robots, it is $+X_{FLANGE}$ that projects from the flange. For these robots, the following applies:

- The methods for orientation calibration (A, B, C) cannot be applied meaningfully to them in most cases.
- **Recommendation:** Enter the C value numerically. A and B must be 0.

In many cases, the axis that projects from the flange can be noted from the documentation of the robot: See the information there about the load center of gravity. Otherwise, contact KUKA.

Use

- Calibration of the tool direction of a tool
- Calibration of the tool direction of a fixed tool

The method **ABC world** is generally used if only the tool direction is to be defined and the other axis directions are irrelevant.

There are 2 variants of this method:

- **5D**: only the tool direction is defined. 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**: with the **6D** variant, it is essentially possible to define all axis directions except the tool direction. This is not as accurate, however, as the **ABC 2-point** method. For this reason, **ABC 2-point** is generally used if all axis directions are to be defined.

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.

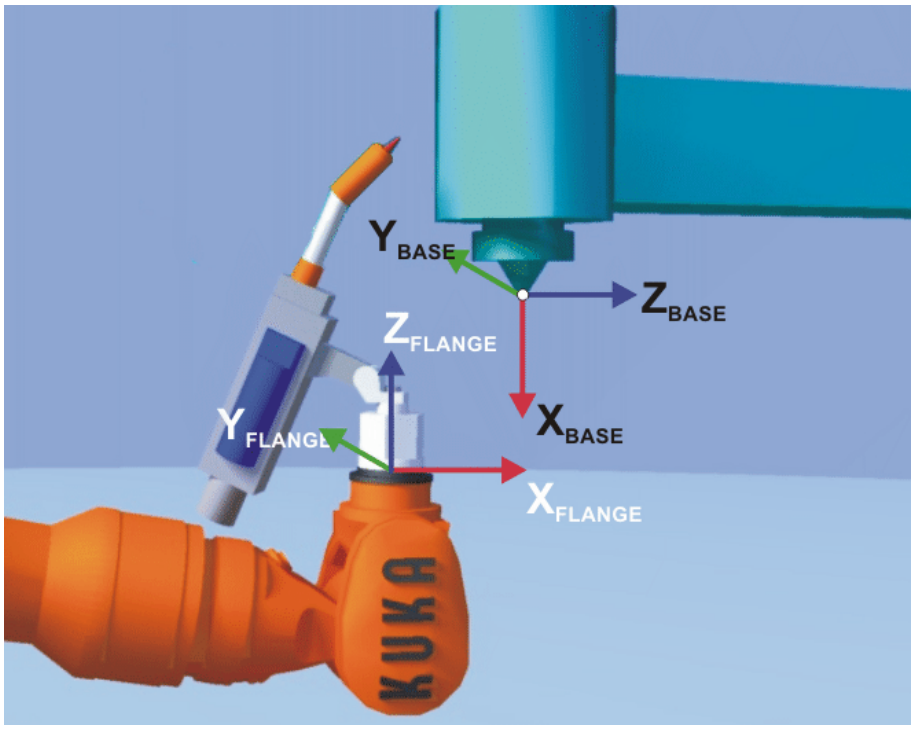


Fig. 5-33: ABC world method for a fixed tool

5.18.9.6 ABC 2-point method

Information about the orientation of certain 4-axis robots
For most robots, **+Z_{FLANGE}** is the axis of the FLANGE coordinate system that projects out of the flange.
With some types of 4-axis robots, it is **+X_{FLANGE}** that projects from the flange. For these robots, the following applies:

- The methods for orientation calibration (A, B, C) cannot be applied meaningfully to them in most cases.
- **Recommendation:** Enter the C value numerically. A and B must be 0.

In many cases, the axis that projects from the flange can be noted from the documentation of the robot: See the information there about the load center of gravity. Otherwise, contact KUKA.

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 all axis directions are to be defined, not just the tool direction.

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

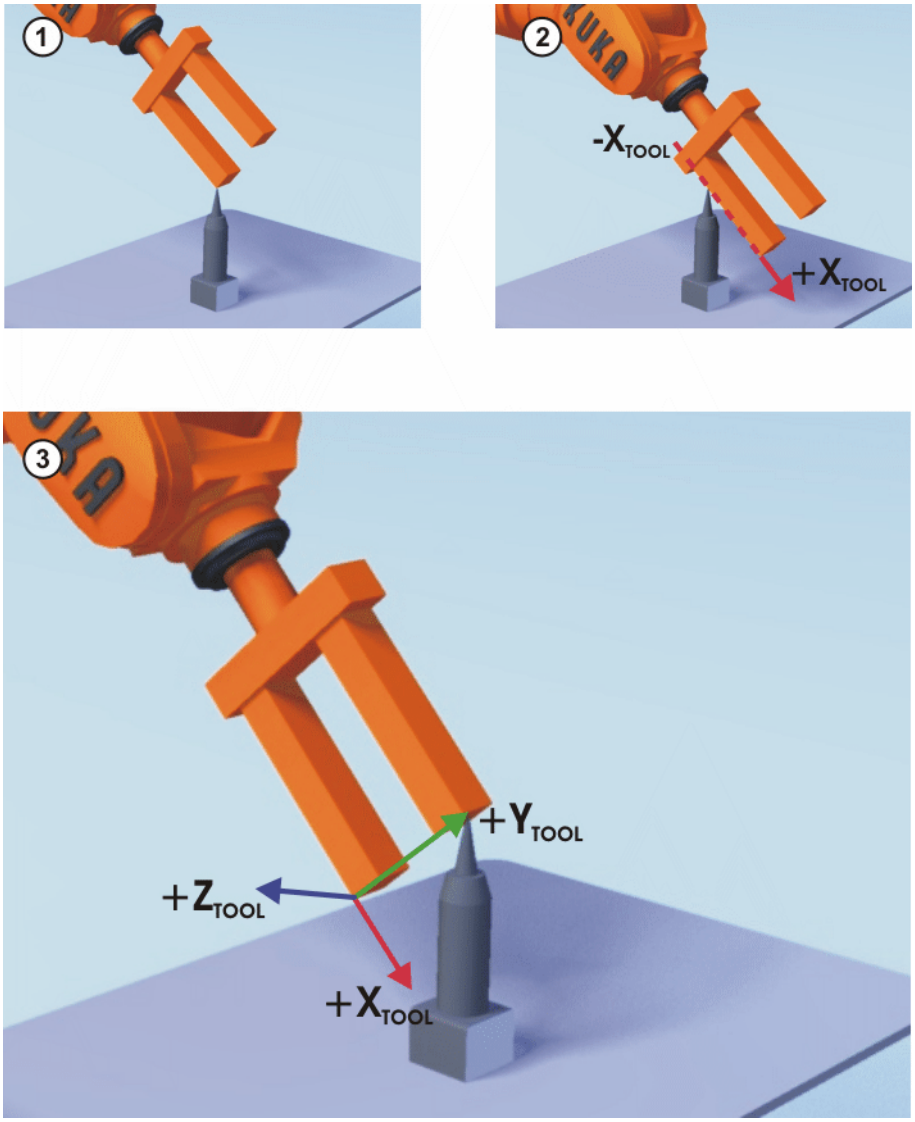


Fig. 5-34: ABC 2-point method for a tool

5.18.9.7 3-point method

Use

- Calibration of X, Y, Z, A, B, C of a 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.
- 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.

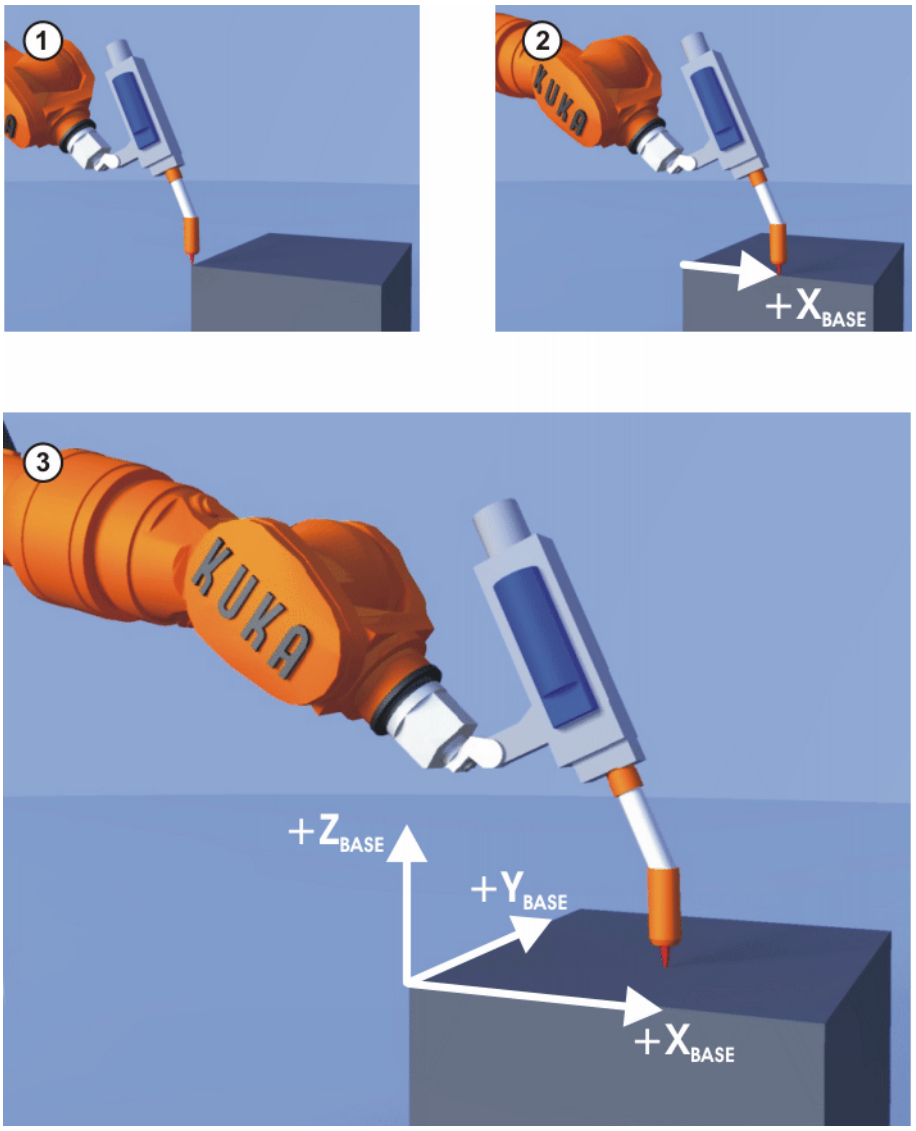


Fig. 5-35: 3-point method for a base

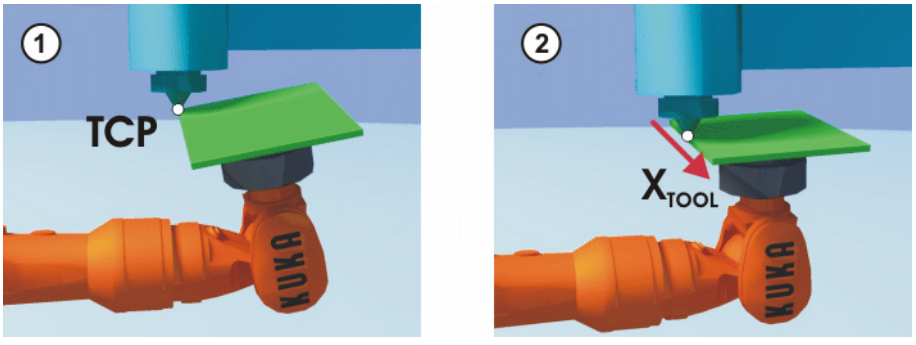


Fig. 5-36

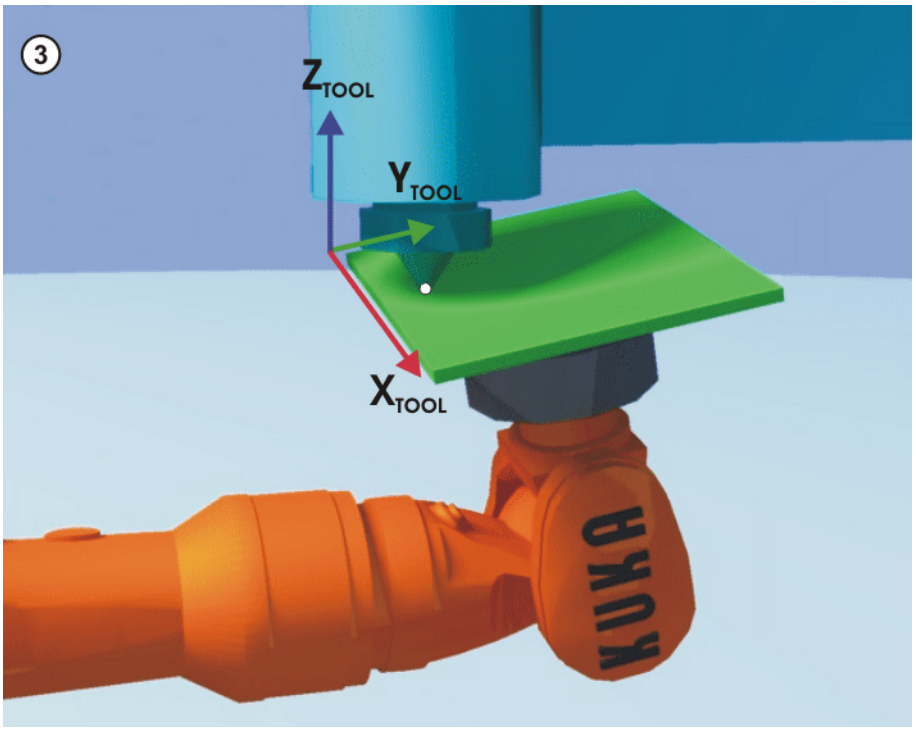


Fig. 5-37: 3-point method for a workpiece on the flange

5.18.9.8 Indirect method

Use

- Calibration of X, Y, Z, A, B, C of a 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.
- Calibration of X, Y, Z, A, B, C of a workpiece on the flange

The **Indirect** method is not 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 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.

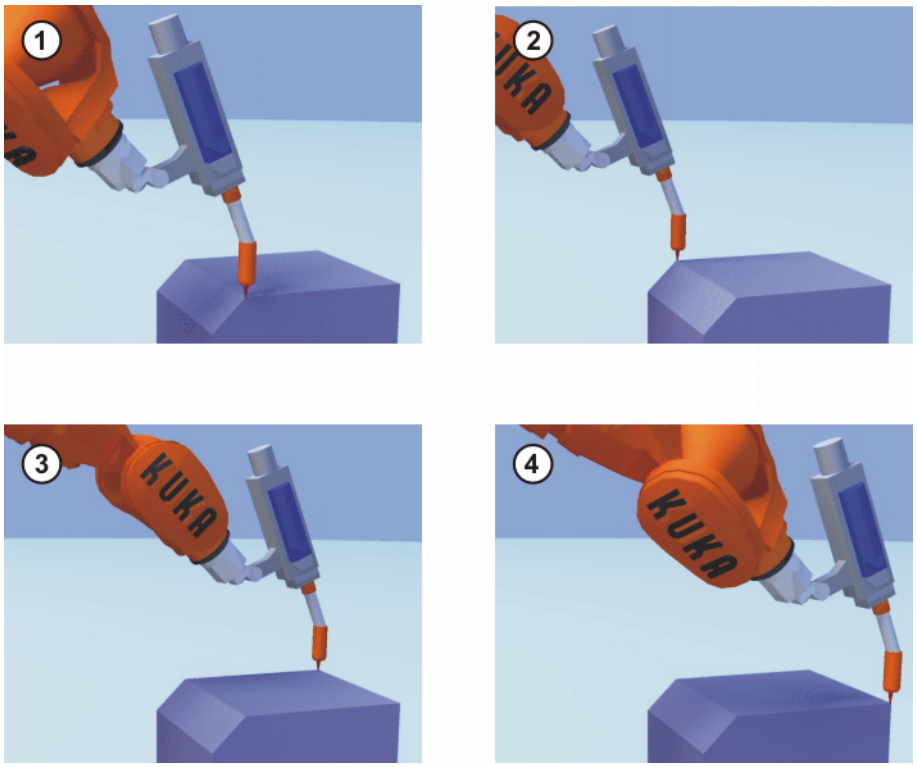


Fig. 5-38: Indirect method for a base

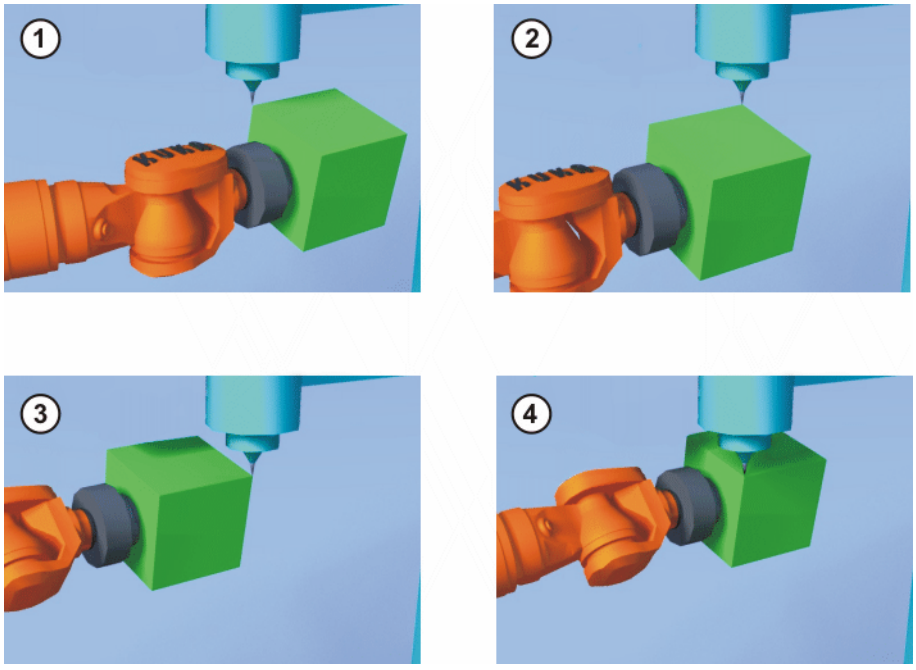


Fig. 5-39: Indirect method for a workpiece on the flange

5.18.9.9 Method Root point

Use

- Calibration of X, Y, Z, A, B, C of the root point of an external kinematic system relative to WORLD

Precondition

- A previously calibrated tool is mounted on the flange.

Description

The TCP of a tool that has already been calibrated is moved to the reference mark on the kinematic system 4 times. The position of the reference mark must be different each time. This is achieved by moving the axes of the kinematic system. The robot controller uses the different positions of the reference mark to calculate the root point of the kinematic system.

External kinematic systems from KUKA have a reference mark. Its position is configured in the system variable $ET*x*_TPINFL in the machine data (*x* = number of the kinematic system).

In the case of non-KUKA external kinematic systems, a reference mark must be determined in advance and configured in the machine data.

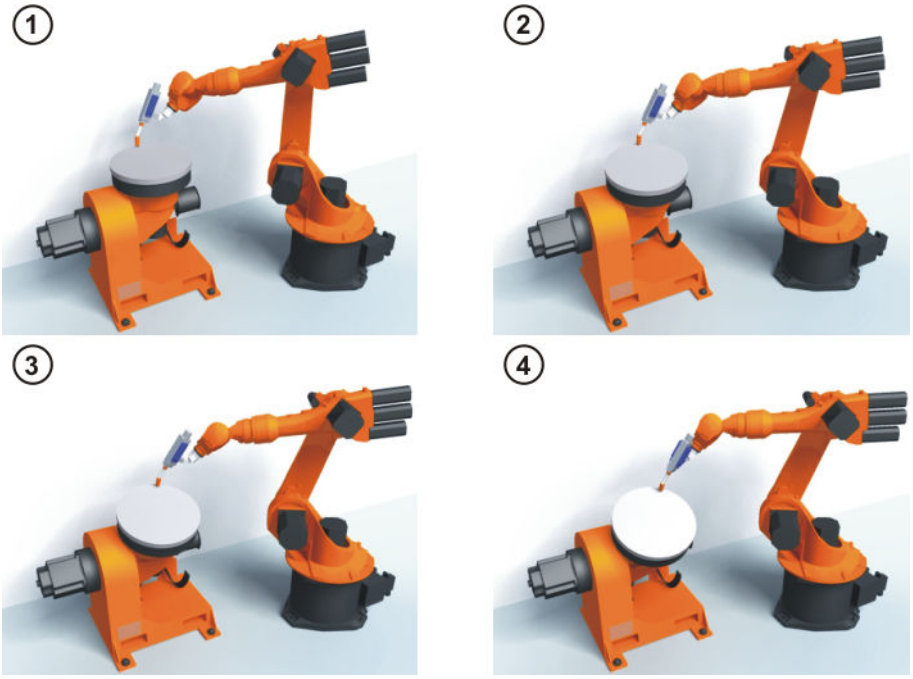


Fig. 5-40: Root point calibration principle

5.18.10 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

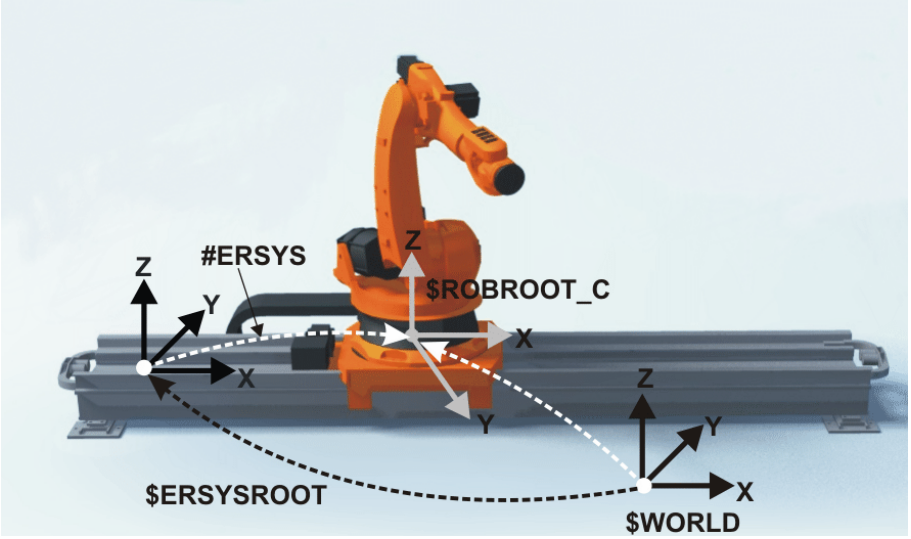


Fig. 5-41: ROBROOT kinematic system – linear unit

5.18.10.1 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 in WorkVisual and loaded into the robot controller.
- A previously calibrated tool is mounted on the mounting flange.
- No program is open or selected.
- T1 mode

In reality, the robot must stand on the linear unit just as it was configured in WorkVisual.

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.

5.18.10.2 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 $ETx_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) ($EX_KIN)
   - **Axis**: number of the external axis (1 … 6) ($ETx_AX)
   - **Name of ext. kinematic system:** ($ETx_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.

5.18.10.3 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.

5.19 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.



Further 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.

5.19.1 Load data parameters

Parameters

Parameter/unit		Description
M	kg	Mass of the load
X, **Y**, **Z**	mm	Position of the center of mass in the reference system
A, **B**, **C**	Degrees	Orientation of the principal inertia axes • A: Rotation about the Z axis of the reference system The result is a coordinate system named 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

Parameter/unit		Description
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°

5.19.2 Entering payload data numerically

WARNING

Danger to life and limb due to incorrect loads
Operating a robot with incorrect loads may result in death, severe injuries or damage to property.

- Use correct load data.
- Use only loads for which the robot is suitable.

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**.
   (>>> *5.19.1 "Load data parameters" Page 214*)

4. If load data verification is available (this depends on the robot type): configure as required.
   (>>> *5.19.5 "Load data verification" Page 219*)
5. Press **Save**.

5.19.3 Entering supplementary load data numerically

WARNING

Danger to life and limb due to incorrect loads
Operating a robot with incorrect loads may result in death, severe injuries or damage to property.
- Use correct load data.
- Use only loads for which the robot is suitable.

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**.
   A list with 3 entries is displayed for A1, A2 and A3.
2. Expand the entries if necessary and enter the load data.
   (>>> *5.19.1 "Load data parameters" Page 214*)
   The load data for A3 can also be imported. (>>> *5.19.4 "Importing supplementary load data for A3" Page 217*)
3. If required, the load data can be reset to the default values for each axis. In each entry, a button is available for this purpose.
   - Default values for A1: “0” for all parameters
   - Default values for A2: “0” for all parameters
   - Default values for A3: dependent on robot type
4. Press **Save**.

Icons in the Supplementary load data window

Icon	Load data
	The load data were entered by the user. In other words, the data were manually entered or imported or applied using the **Default values** button.
	The **Mass** box contains the value “-1”. The correct load data must still be entered.

5.19.4 Importing supplementary load data for A3

WARNING

Danger to life and limb due to incorrect loads
Operating a robot with incorrect loads may result in death, severe injuries or damage to property.

- Use correct load data.
- Use only loads for which the robot is suitable.

Description

The energy supply system is normally located on A3. The supplementary load data for energy supply systems can be imported.

- For many KUKA robot types, the controller automatically provides the data for common energy supply systems.
  The user can select an energy supply system and import the data.
- It is also possible to import the load data for user-defined energy supply systems.

Preparation

Only necessary if load data of user-defined energy supply systems are to be imported:

1. Create an XML file with the data of the energy supply system(s), either via KUKA.Load Pro or manually.
   One XML file may contain data of multiple energy supply systems for various robot types. If the file is created manually, it must have the same structure as a file from KUKA.Load Pro.

   More information is contained in the KUKA.Load/KUKA.Load Pro documentation.

2. Create the **Additional_Load** folder on the controller under C:_KRC \USER. Paste the XML file into the folder.
   The data are immediately available for import.

   The **Additional_Load** folder may only contain one single XML file.

Precondition

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

Procedure

1. In the main menu, select **Start-up** > **Supplementary load data**.
   The **Supplementary load data** window opens.
2. In the area **Axis 3**:
   a. Filter the types if required: to do so, enter a character string in the **Filter** box.
      Upper and lower case are distinguished!
   b. Select the application in the **Application** box.
      If only one application is available, it is selected automatically.
      If a filter is set, only applications with matching types are available for selection.
   c. Select the energy supply system in the **Type** box.

Only energy supply systems that match the current robot, the selected application and, if applicable, the filter criterion are available for selection.

The load data are determined and entered automatically.

3. Press **Save**.

If required, the load data can be reset using the **Default values** button. The default values applicable to A3 depend on the robot type.

“Supplementary load data” window

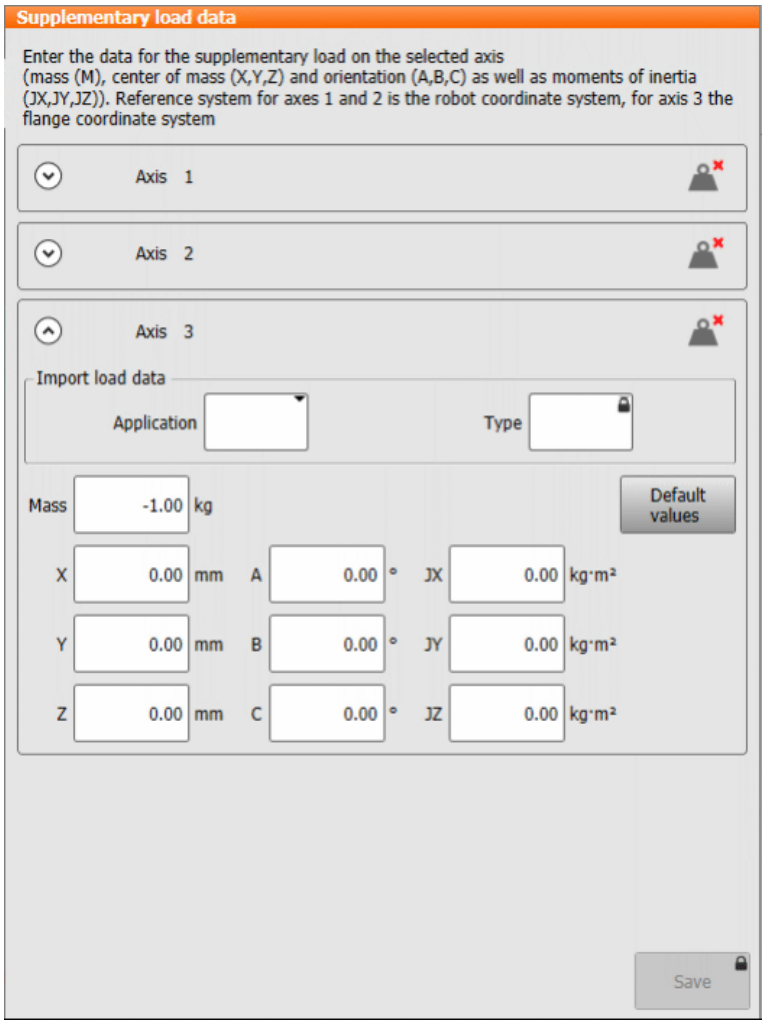


Fig. 5-42: Supplementary load data window with imported values on A3

Icons in the Supplementary load data window

Icon	Load data
	The load data were entered by the user. In other words, the data were manually entered or imported or applied using the **Default values** button.
	The **Mass** box contains the value “-1”. The correct load data must still be entered.

5.19.5 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 220*)

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**

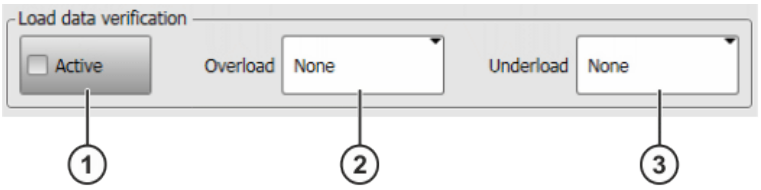


Fig. 5-43: Load data verification

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 =` *Result*

Element	Description
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.

5.20 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

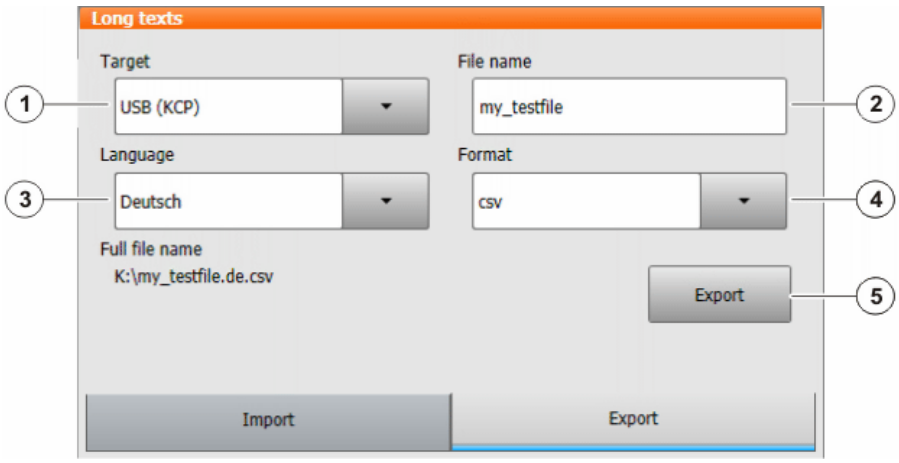


Fig. 5-44: Exporting long texts

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

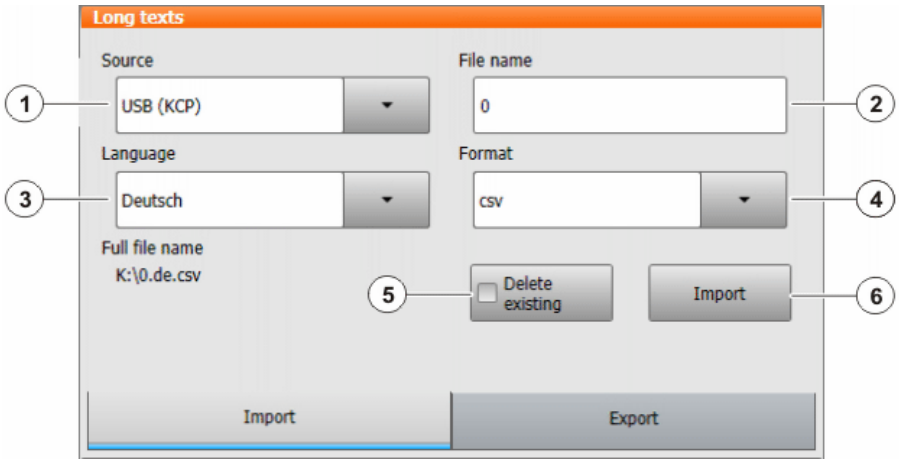


Fig. 5-45: Importing long texts

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 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.

Item	Description
5	• **Active:** All existing long texts are deleted. The contents of the file are applied. • **Inactive:** 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.

5.21 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**

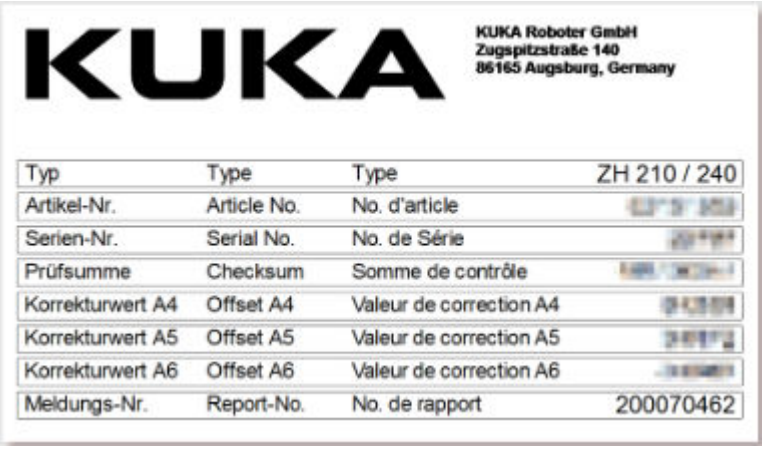


Fig. 5-46: In-line wrist rating plate: example in German, English, French

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.

5.22 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.

5.22.1 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.

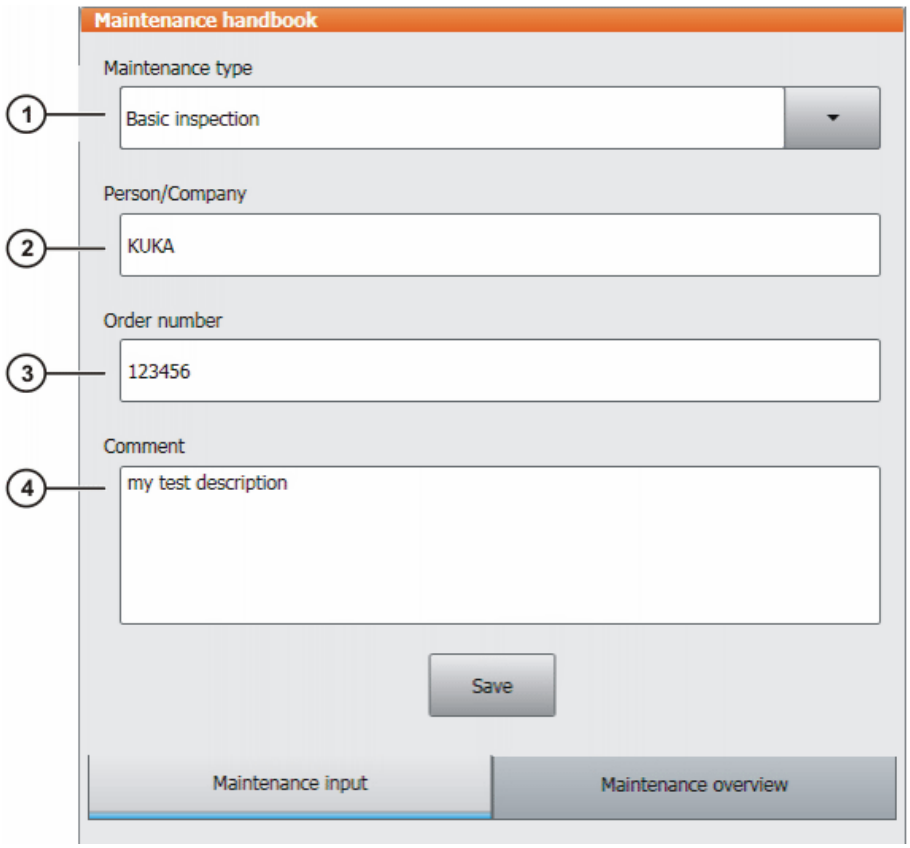


Fig. 5-47: Maintenance input

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.

5.22.2 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.

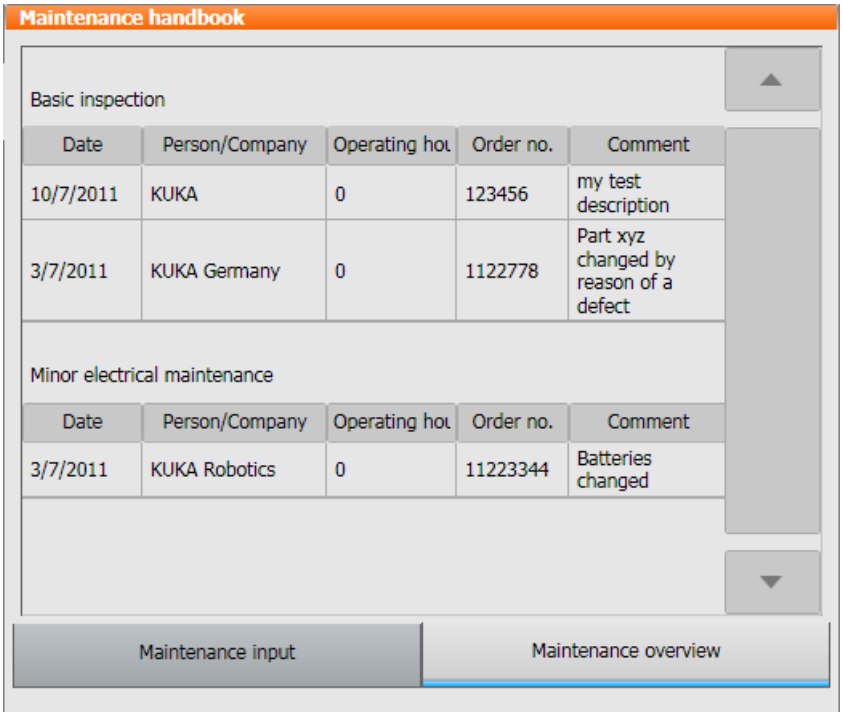


Fig. 5-48: Maintenance overview

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:

- PROFINET
- 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 236*)

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.

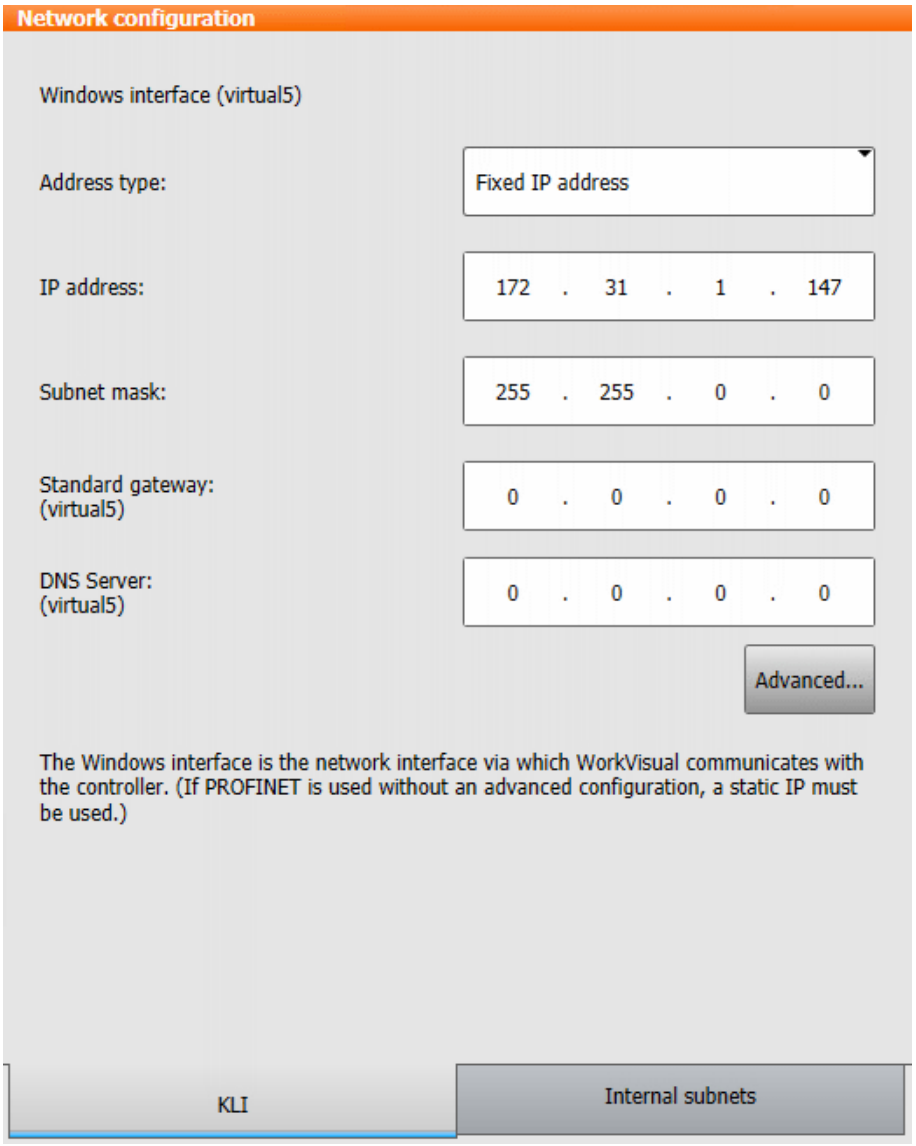


Fig. 6-1: Example: Fixed IP address

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.

	A shared DNS server address and a shared default gateway can be used for the interfaces.

Precondition

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

- If KUKA.PROFINET is installed: the PROFINET device has been named.

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 236*)

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 **Fixed IP address** type 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.

   Various reception filters are now displayed in the right-hand column.
8. The lower (!) **Reception filter:** box indicates **Accept all**.

   Change the entry to **Target IP address**.

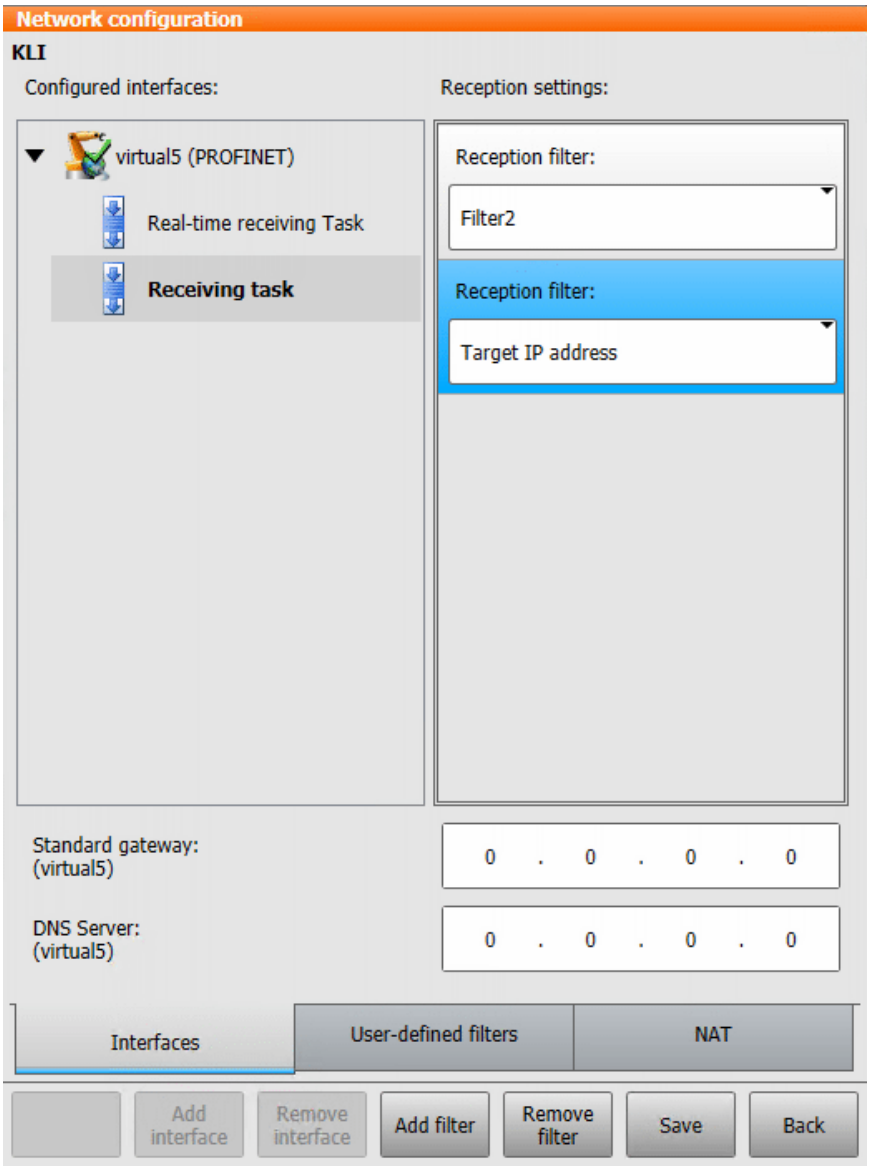


Fig. 6-2: Reception filter, with “PROFINET” as example

9. Do not change any other reception filters.
10. Select the entry “virtual5” in the left-hand column and press the **Add interface** button. The entry “virtual6” is automatically created.
11. Select the entry “virtual6” and enter “WINDOWS” in the **Interface designation:** box.
12. 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”.

13. Set the check mark in the check box **Windows interface**.

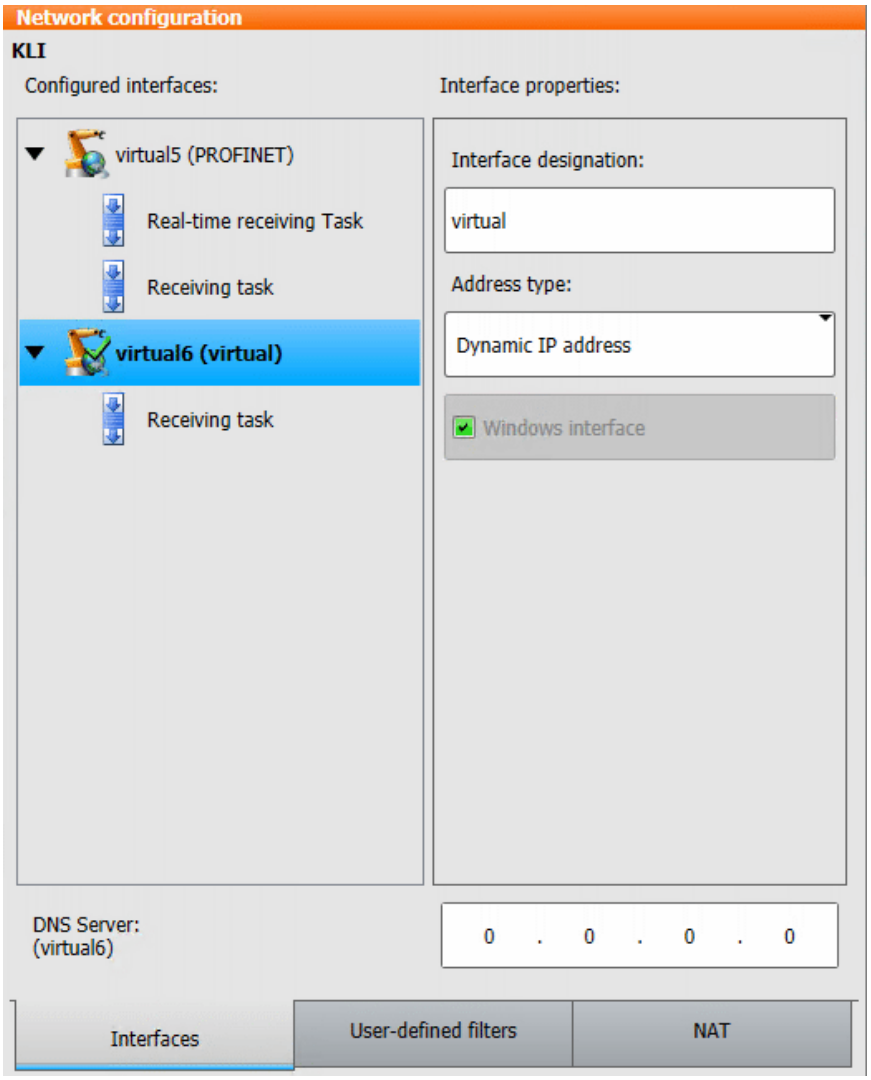


Fig. 6-3: “virtual6” interface, with “WINDOWS” as example

14. Select the “Reception task” entry under the “virtual6” entry.
15. If not already the case, change the **Reception filter:** box in the right-hand column to **Accept all**.
16. Press **Save**.
17. Close the **Network configuration** window using the Close icon.
18. 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

Loss of functions due to removal of ports
If the ports that are enabled as standard by KUKA are removed, this may result in the loss of functions of the robot controller.

- Do not remove the ports that are enabled as standard by KUKA.

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

Loss of functions due to modification of subnet configuration
Modifications of the subnet configuration of the robot controller may result in the loss of functions of the robot controller.

- Only modify the subnet configuration after consultation with KUKA.

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

Loss of functions due to modification of subnet configuration
Modifications of the subnet configuration of the robot controller may result in the loss of functions of the robot controller.

- Only modify the subnet configuration after consultation with KUKA.

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 236*)

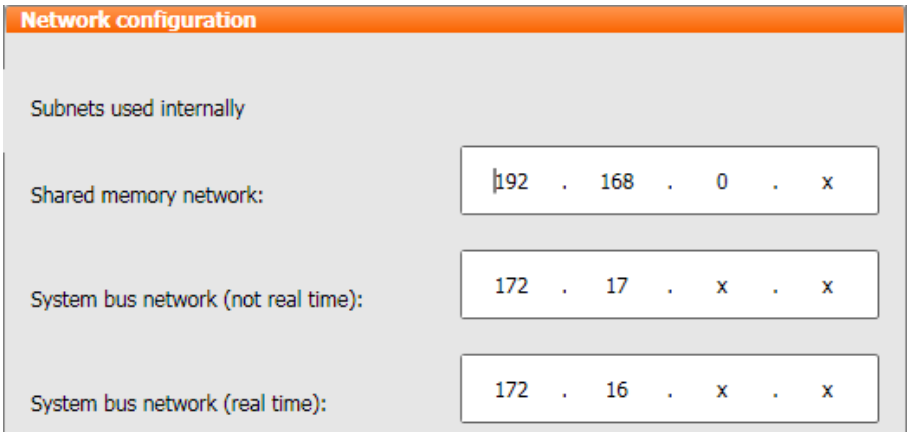


Fig. 6-4: Example: Internal subnets

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 coupable 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

Danger to life and limb due to failure to check the safety configuration
After the safety configuration has been modified, it may contain incorrect data in exceptional cases. If it is not checked and, if necessary, corrected, death to persons, severe injuries or considerable damage to property may result.

- Following modifications to the safety configuration, always check the values for the safe axis monitoring functions.
  (>>> *6.5 "Checking the values for the safe axis monitoring functions" Page 244*)

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 242*)

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 240*)
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 242*)
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, or after the configured braking time at the latest.

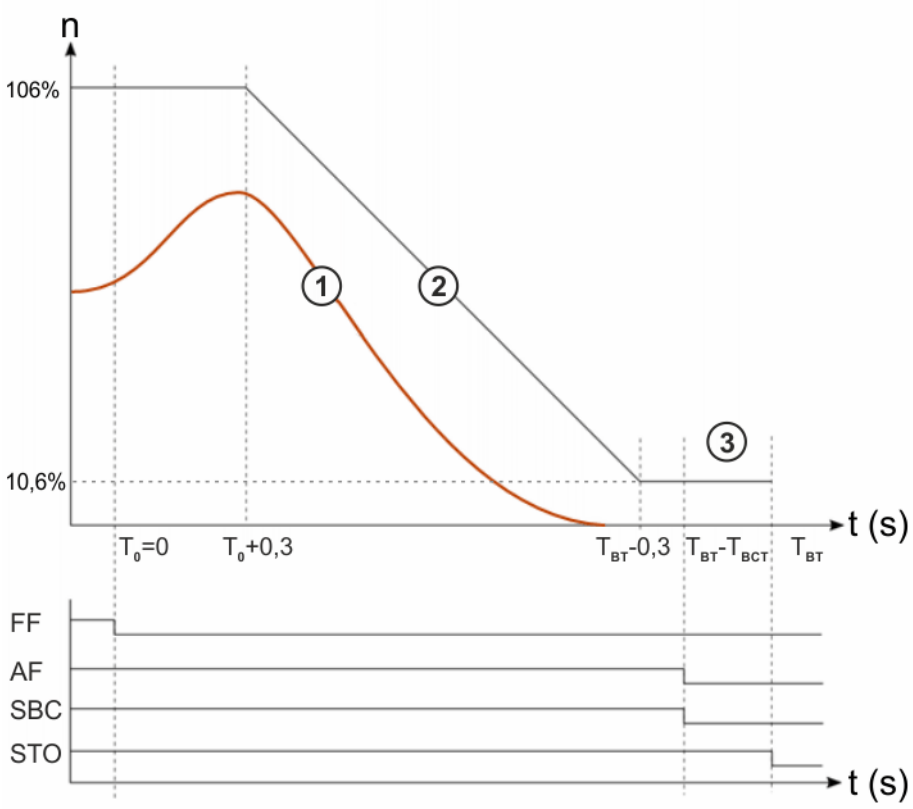


Fig. 6-5: Monitoring ramp for safety stop 1

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.

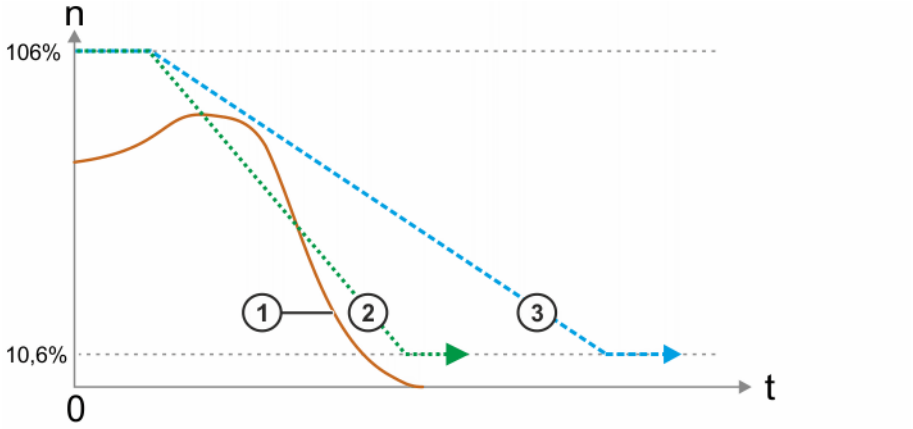


Fig. 6-6: Example: Value is increased

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 243*)
   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

Danger to life and limb due to failure to check “Braking time” and “Position tolerance”
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.

- Always check the values if the activation code on the **Common** tab in the **Safety configuration** window has changed.

  In other words, not only if the values themselves have been changed, but if any changes have been made that affect the safety configuration.

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

Danger to life and limb due to unchecked safety configuration
If the safety configuration is not checked and updated where necessary, it may contain incorrect data. Death, serious injuries or major damage to property may result.

- After activation of a WorkVisual project on the robot controller and changes to the machine data, check the safety configuration on the robot controller.

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 245*)
   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 Activation code of the safety configuration

Description

The activation code is updated each time the safety configuration is saved. The robot controller displays the activation code 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
  Also in the file EXP_SCTLCRC.xml in the directory C:\KRC\ROBOTER\LOG\Safety (provided that a project has already been activated or the safety configuration has been saved)

The file SCTLCRC.xml is available so that, if required, the system integrator can read the activation code 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 ActivationCode="49CCA230" />
```

Fig. 6-7: Example: activation code 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.

Automatic export

The robot controller automatically exports the safety configuration in the following cases:

- Immediately after every project activation
- When saving the safety configuration

The scope of the automatic export corresponds to the manual export described above via **Configuration** > **Safety configuration** > **Export**.

The automatic export creates the following files in the directory C:\KRC\ROBOTER\LOG\Safety:

- EXP_SCTLCONFIG.xml: Contains the safety parameters
- EXP_SCTLCRC.xml: Contains the activation code of the safety configuration
- EXP_SCLOG.xml: Contains the change log of the safety configuration

6.8.1 Changing the directory for the export

Description

If the safety configuration is exported automatically, then the export is always saved in the directory **C:\KRC\ROBOTER\LOG\Safety**.

The directory can be altered in order to save the automatic export in a specific location.

Configuration file

Directory	C:\KRC\USER
File	SmartHMI.User.config

Procedure

1. Open Windows Explorer or the Navigator.
2. Navigate to the directory in which the file is located.
3. Open the file.
4. Search for `AutomaticXmlExportPath` and enter the desired directory as a value.
5. Save and close the file.

6.8.2 Switching off automatic export

Description

If an automatic export of the safety configuration is not required, then the automatic export can be switched off.

Configuration file

Directory	C:\KRC\USER
File	SmartHMI.User.config

Procedure

1. Open Windows Explorer or the Navigator.
2. Navigate to the directory in which the file is located.
3. Open the file.
4. Search for `AutomaticXmlExportEnabled` and set the value to **false**.
5. Save and close the file.

6.9 Protecting a digital output against manual modifications

Description

Digital outputs can be protected against manual modifications. Their state (TRUE/FALSE) can then no longer be changed in the **Digital I/O** window. The name can still be changed there.

The state of the output can still be changed via the variable display and via a program.

Preparation

The required configuration file **kImdigitalio.WriteProtectedOutputs.cfg** is not available on the robot controller from the outset. It is created automatically when the **Digital I/O** window is opened for the first time.

- If necessary, open the window via **Display** > **Inputs/outputs** > **Digital outputs** (or **Digital inputs**).

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. Open the file **kImdigitalio.WriteProtectedOutputs.cfg** at Windows level in C:\KRC\USER.
   As standard, the file contains the line `<int>0<int>`. In other words, no output is protected.
3. Instead of “0”, enter the number of the desired output.
   For further outputs, add more `<int>…<int>` lines.

```
klmdigitalio.WriteProtectedOutputs.cfg
1  <?xml version="1.0" encoding="utf-8"?>
2  <ArrayOfInt xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3    <int>30</int>
4    <int>31</int>
5  </ArrayOfInt>
```

Fig. 6-8: Example: Protecting digital outputs 30 and 31

4. Save and close the file.
5. To maximize the smartHMI again, touch the **smartHMI** icon in the task bar:



6. Reboot the robot controller. The robot controller automatically carries out a cold restart.

In the **Digital I/O** window, the outputs are now labeled “RO” (= read only).

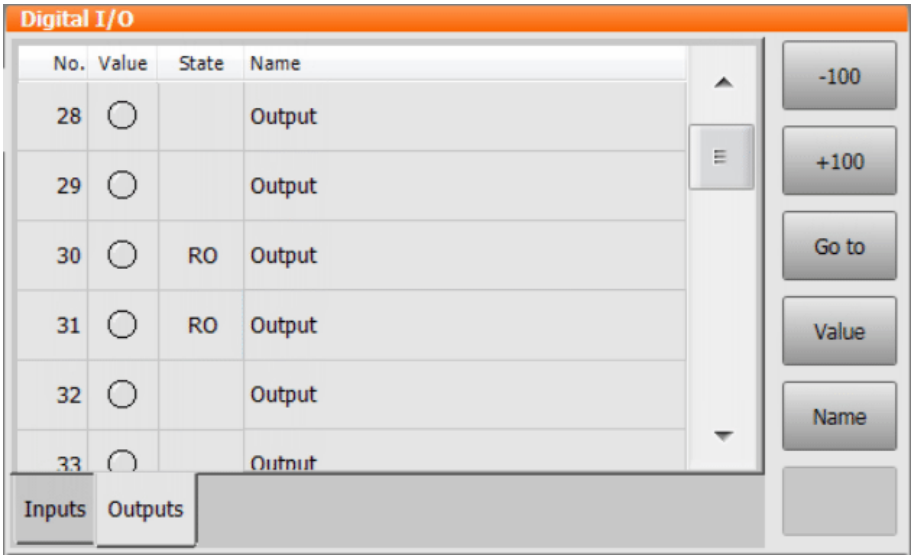


Fig. 6-9: Example: Digital outputs 30 and 31 are protected

6.10 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

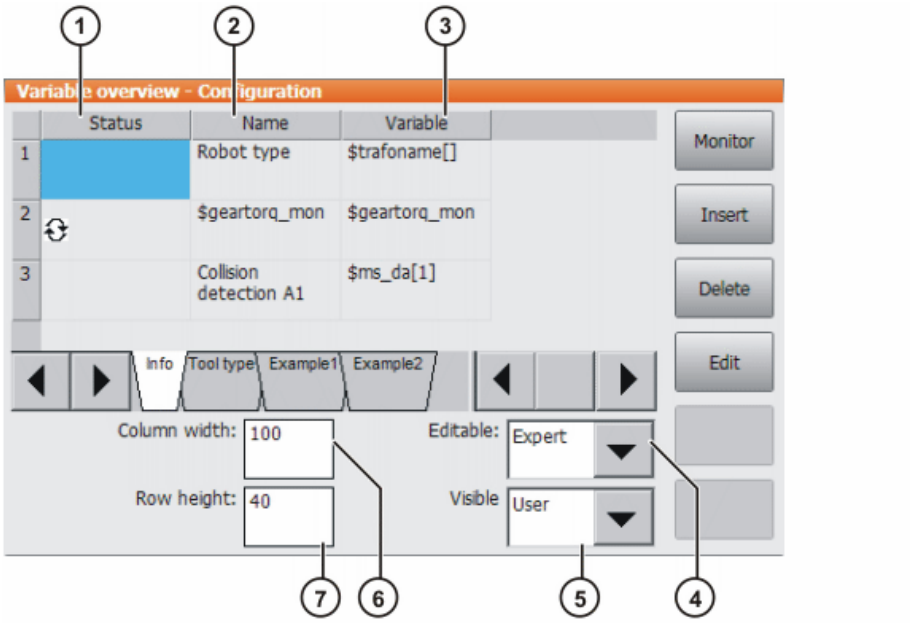


Fig. 6-10: Variable overview - Configuration

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. (>>> *4.20.7 "Displaying the variable overview and modifying variables" Page 114*)
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.

6.11 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.

6.12 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.
(>>> *6.12.2 "Overview of function groups" Page 253*)

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.

6.12.1 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* […]"

6.12.2 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/modify 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**
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 Tech packages	**Basic operation of the installed technology packages, e.g. pressing a status key** Default setting: **User**
Configuration of Tech packages	**Configuration of the installed technology packages, e.g. settings in the configuration plug-in** Default setting: **Expert**
Advanced configuration of Tech packages	**Advanced configuration of the installed technology packages, particularly critical settings** Default setting: **Administrator**

6.13 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.

6.14 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 ECO_LEVEL Degree of energy saving • #OFF: energy saving mode is deactivated. • #LOW: low saving • #MIDDLE: medium saving • #HIGH: high saving

6.15 Configuring workspaces

WARNING

Danger to life and limb due to missing personnel protection
These workspaces serve to protect the system. They are not monitored using safe technology and do not serve to protect personnel. Failure to observe this may result in death, severe injuries or damage to property.

- Ensure personnel protection using appropriate means.

These workspaces are part of the System Software and are independent of workspaces in KUKA.SafeOperation or other safety options.

Workspaces can be configured for a robot. 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, but several more finely differentiated modes.

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

6.15.1 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

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

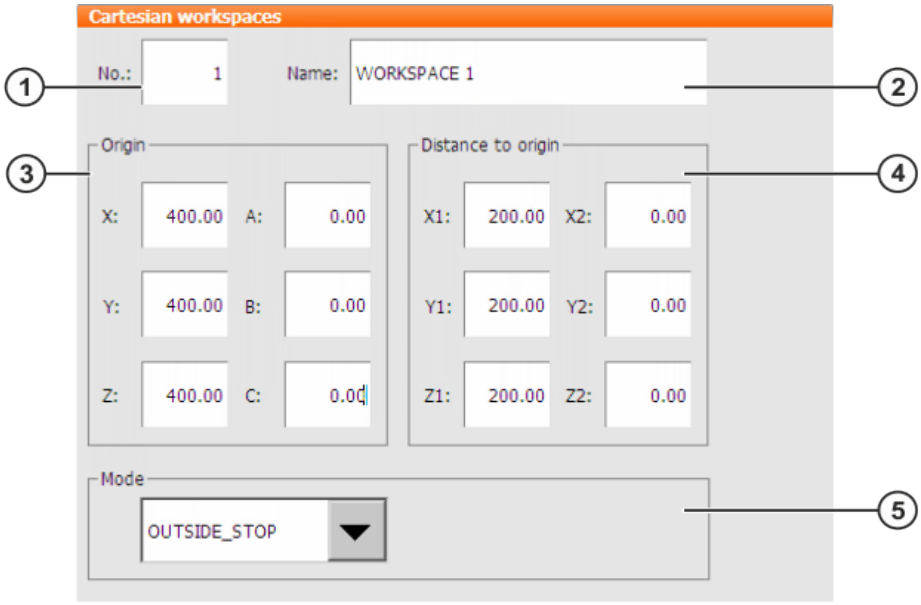


Fig. 6-11: “Cartesian workspaces” window

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	Extension of the workspace in mm relative to the origin X1, Y1, Z1 define a cuboid. • X1, Y1, Z1 without sign (corresponds to positive sign): The cuboid extends from the origin in the plus direction. • X1, Y1, Z1 with negative sign: The cuboid extends from the origin in the minus direction. X2, Y2, Z2 can expand or shrink the cuboid spanned by X1, Y1, Z1. • X2 with a different sign to X1: The cuboid edge that is on the X-axis becomes longer. • X2 has the same sign as X1: The cuboid edge that is on the X-axis becomes shorter. • Applies in the same way to the coordinate pairs “Y1, Y2” and “Z1, Z2”. If, for at least one pair of coordinates “X1, X2”, “Y1, Y2” and “Z1, Z2”, both values have the same sign, then the origin will be outside the workspace.
5	Mode (>>> *6.15.2 "Modes for Cartesian workspaces" Page 262*)

(>>> *"Example 1" Page 260*)

(>>> *"Example 2" Page 261*)

(>>> *"Example 3" Page 262*)

“Signals” window

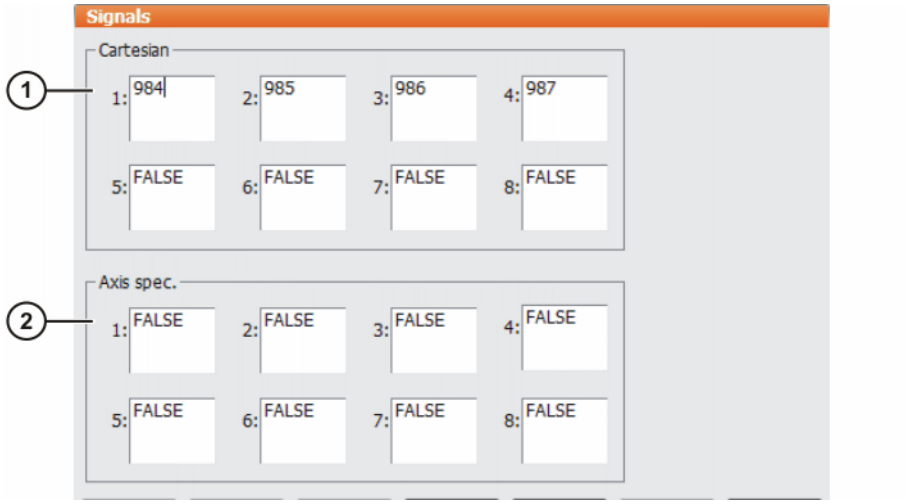


Fig. 6-12: “Signals” window

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.

Example 1

The example shows a simple workspace.

Figure (>>> *Fig. 6-11*) shows the settings that can be used to create such a workspace.

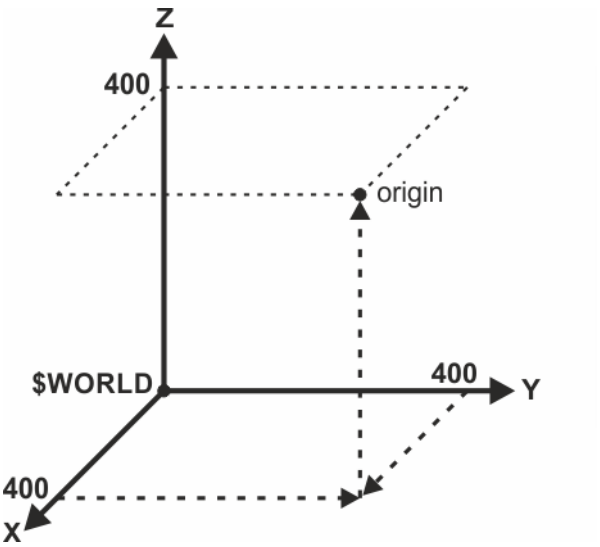


Fig. 6-13: Cartesian workspace, “origin”

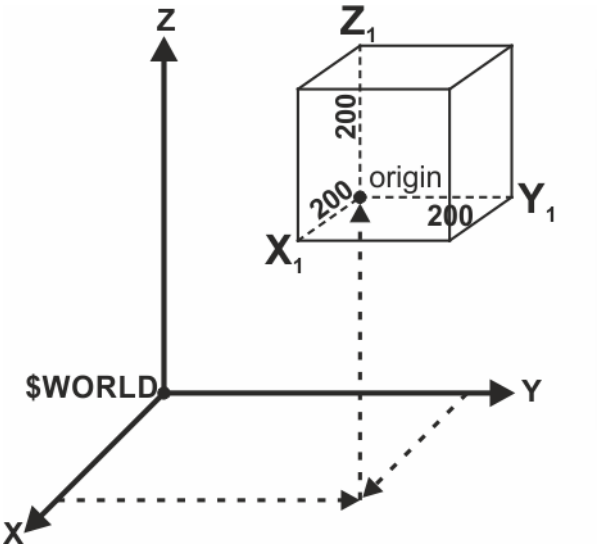


Fig. 6-14: Cartesian workspace, dimensions

Example 2

X1, Y1, Z1 define a cuboid.

Y2 reduces the size of the cuboid because it has the same sign as Y1.

Because the cuboid has been reduced in size, the origin is now outside the cuboid.

The table below shows the settings that can be used to create such a workspace.

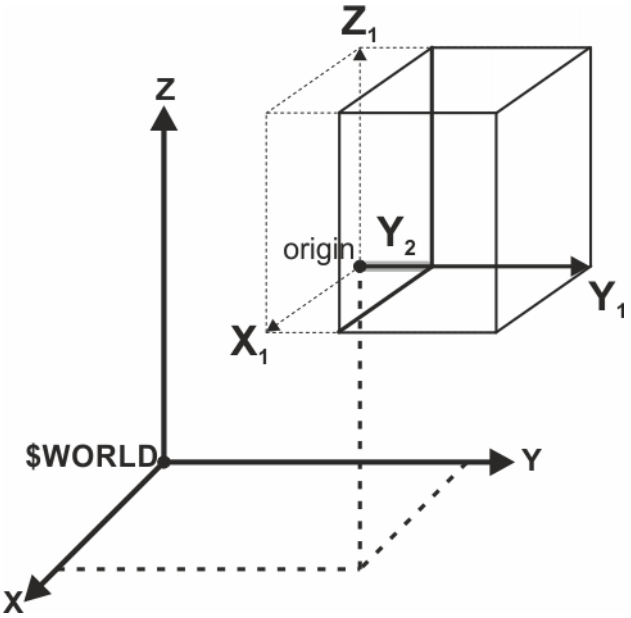


Fig. 6-15: Cartesian workspace, dimensions: Example 2

Settings in the **Cartesian workspaces** window:

Origin				**Distance to origin**			
X:	400.00	**A:**	0.00	**X1:**	300.00	**X2:**	0
Y:	400.00	**B:**	0.00	**Y1:**	300.00	**Y2:**	100.00
Z:	400.00	**C:**	0.00	**Z1:**	300.00	**Z2:**	0

Example 3

X1, Y1, Z1 define a cuboid.

X2, Y2, Z2 expand the cuboid because they have a different sign from X1, Y1 and Z1.

Because the cuboid has been expanded in all directions, the origin is within the cuboid.

The workspace is also tilted around Y_{WORLD} and around Z_{WORLD}.

The table below shows the (approximate) settings that can be used to create such a workspace.

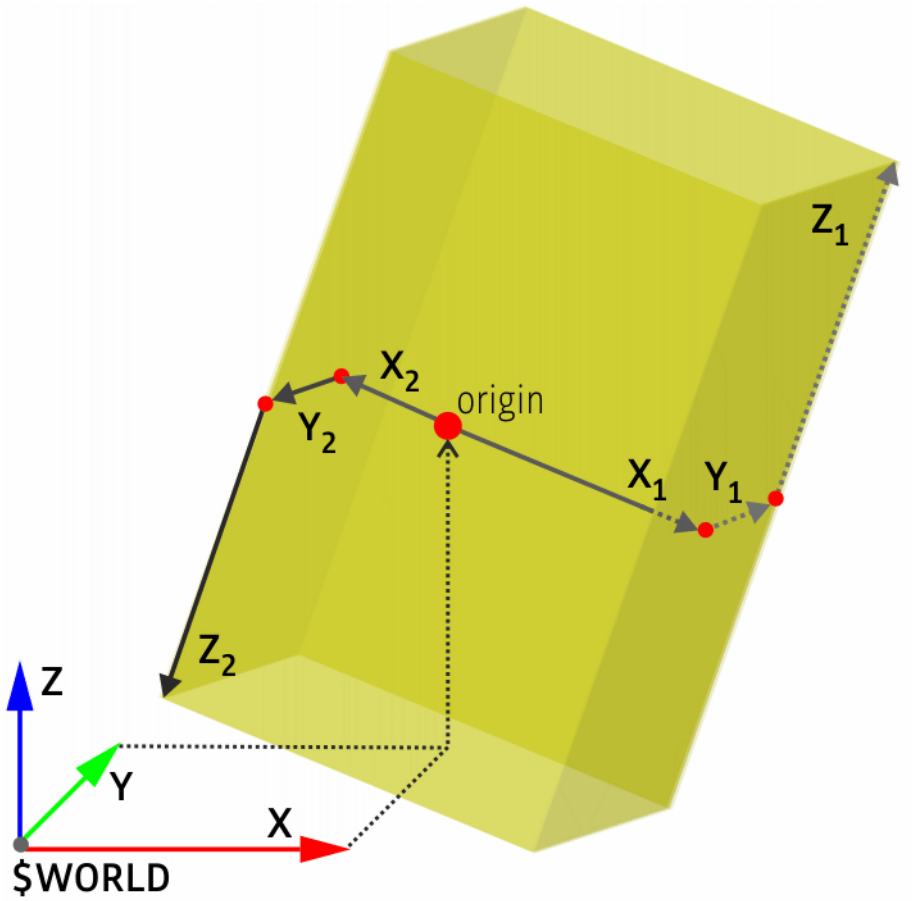


Fig. 6-16: Cartesian workspace, dimensions: Example 3

Settings in the **Cartesian workspaces** window:

Origin				Distance to origin			
X:	410	**A:**	30.00	**X1:**	370.00	**X2:**	-160.00
Y:	200	**B:**	20.00	**Y1:**	90.00	**Y2:**	-110.00
Z:	390	**C:**	0.00	**Z1:**	480.00	**Z2:**	-400.00

6.15.2 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.

Mode box	Description
#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.
#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.
(>>> *4.19 "Bypassing workspace monitoring" Page 105*)	

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.

6.15.3 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.

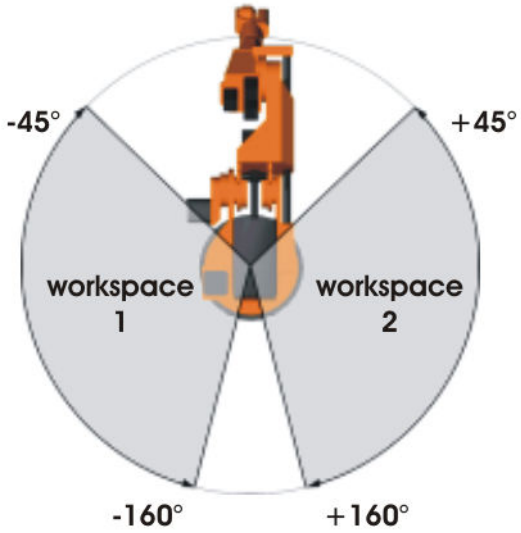


Fig. 6-17: Example of axis-specific workspaces for A1

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. 6-18: “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 (>>> *6.15.4 "Modes for axis-specific workspaces" Page 266*)

“Signals” window

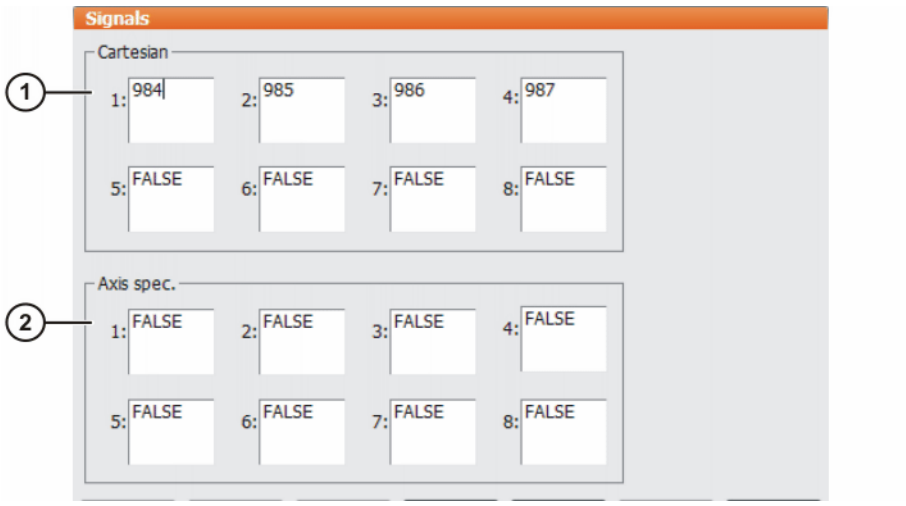


Fig. 6-19: “Signals” window

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.

6.15.4 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.

(>>> *4.19 "Bypassing workspace monitoring" Page 105*)

6.16 Workspace for the KR DELTA

Description

For the KR DELTA product family, there is a predefined Cartesian workspace that the robot must not leave.

- The workspace is automatically present and active if the machine data of a KR DELTA have been loaded. The flange center point is monitored.
- The workspace cannot be deactivated or changed by the user. The workspace takes into consideration the special kinematic system of the KR DELTA, which differs greatly from that of common 6-axis robots.

The possibility of creating user-defined workspaces is always available and is independent of the workspace of the KR DELTA.
(>>> *6.15 "Configuring workspaces" Page 257*)

WARNING

Danger to life and limb due to missing personnel protection
The workspace for the KR DELTA serves to protect the system. It is not monitored using safe technology and does not serve to protect personnel. Failure to observe this may result in death, severe injuries or damage to property.

- Ensure personnel protection using appropriate means.

Response

- The robot stops as soon as the command position leaves the workspace. Messages:
  - *Stop due to workspace violation*
    (Acknowledgement message after every workspace violation)
  - *DELTA workspace violated*
    (Status message additionally for workspace of KR DELTA)
- If the end position or a spline segment point is located outside the workspace, the robot controller does not start the corresponding motion and generates the following acknowledgement message:
  - *Target point {Target point name} outside of DELTA workspace*

  The programmer has the option of skipping the motion command and the message with ON_ERROR_PROCEED. (>>> *11.8.8 "ON_ERROR_PROCEED: skipping an instruction that causes an interpreter error" Page 599*)

Resuming motion after a stop:

- In T1/T2, the user must acknowledge the message *Stop due to workspace violation*. He can then move the robot out of the invalid range.
- In the other operating modes, the robot cannot be moved as long as it is in the invalid range. Status message: *Automatic mode blocked.*

  The user must switch to T1/T2 mode in order to be able to move the robot.
- While the user is moving the robot, the controller monitors whether it is moving towards the valid range. If it is moving away from the valid range, an error stop occurs. The robot controller generates the following acknowledgement message:
  - *DELTA workspace retraction not possible (direction of motion)*

Shape and position

The workspace has the shape of a cylinder on an inverted, truncated cone. In the case of the KR DELTA, the Z axis of $ROBROOT points upwards. The workspace is in the negative Z range.

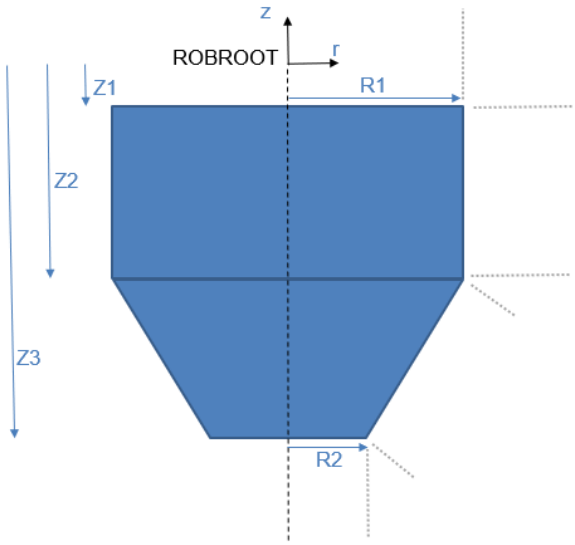


Fig. 6-20: Workspace for the KR DELTA

6.17 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 exceeded, 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/modify 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.

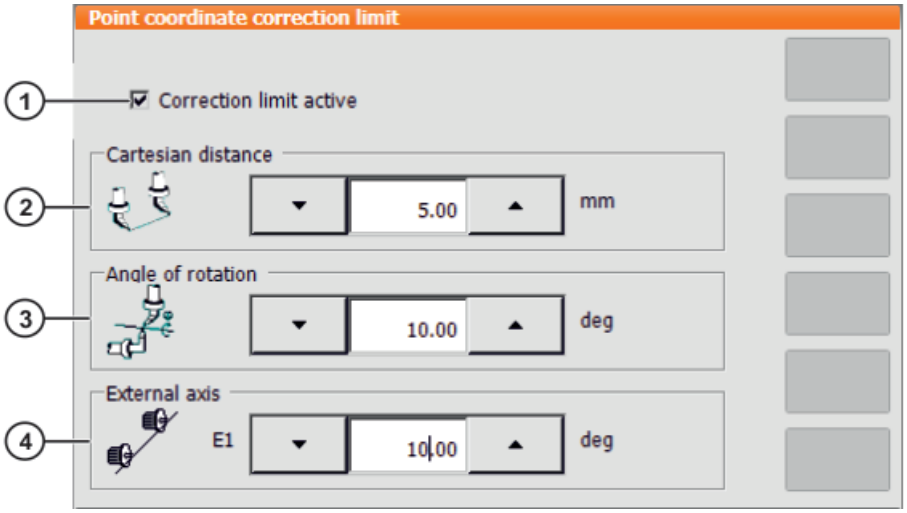


Fig. 6-21: Option window “Point coordinate correction limit”

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.

Item	Description
3	Maximum permissible change for the A, B and C values • **0.00 … 20.00 deg** Value 0.00 means: no change allowed.
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.

6.18 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 *Ackn. Max. following error exceeded ({Drive})*..

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 the following motions:

- PTP motions
- Approximate positioning blocks from a PTP motion to a LIN or CIRC spline motion or vice versa
- Individual SPTP motions and SPTP spline blocks
- Approximate positioning segments from a CP spline block to a PTP spline block

Other motions are not monitored and their velocity is not reduced.

6.18.1 Warm-up sequence

Precondition

- $WARMUP_RED_VEL = TRUE
- Operating mode T2, AUT or AUT EXT
- The robot is considered to be cold. This applies in the following cases:
  - After a cold start, as long as it is under servo control less than $WARMUP_TIME
  - 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.

6.18.2 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) • 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%
$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

6.19 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

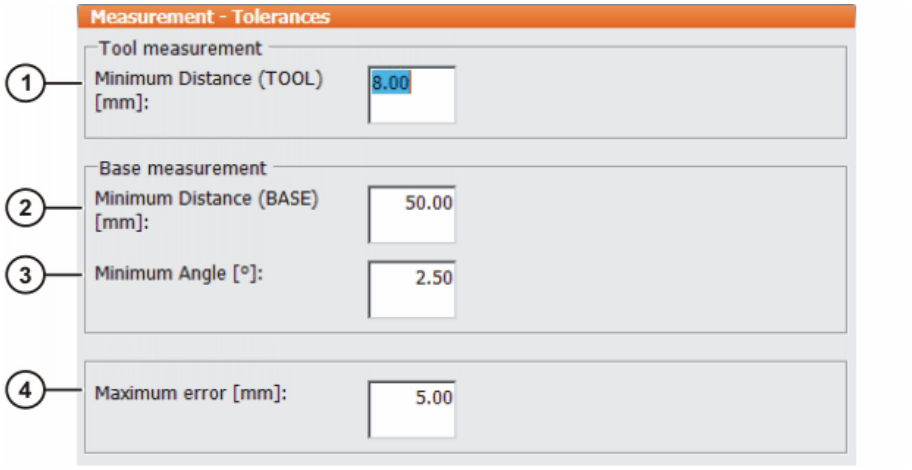


Fig. 6-22: Default error tolerances

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**.

6.20 Configuring “user list” with variables for inline forms

Description

The **User list**button is available in the **WAIT FOR** inline form: The button can be used to display a list of user-defined variables when programming the condition. The list must be configured beforehand.

In KSS 8.6 and higher, the user list is also available in the following inline forms:

- **IF ... THEN**, **WHILE ... ENDWHILE**, **REPEAT ... UNTIL**, **Interrupt declaration**

The list is valid for all of these inline forms. It is not possible to configure a specific list for each inline form.

In other KSS versions, the user list may possibly be available in other inline forms. Whether the user list is available can be found in the description of the individual inline form.

Precondition

- “Expert” user group
- T1, T2 or AUT mode
- No program is selected.

Procedure

1. Create and initialize the required variables in the file $config.dat under KRC:\R1\System:

   In the section “User-defined variables” in the fold “USER GLOBALS”
2. Enter the variable names in the file WaitForUserList.xml under C:\KRC\USER.

   There are example lines for this purpose. They can be adapted and uncommented.

   The upper and lower-case letters of the variable names may vary between $config.dat and WaitForUserList.xml.
   The spelling in WaitForUserList.xml is applied in the inline form list.
3. Carry out a cold start.

(>>> *6.20.1 "Example of a “user list”" Page 273*)

6.20.1 Example of a “user list”

$config.dat

Create and initialize variables in KRC:\R1\System\$config.dat:

Create the variables in the section “User-defined variables” in the fold “USER GLOBALS”.

```
DEFDAT  $CONFIG
...
; Variables for user list
; C:\KRC\USER\WaitForUserList.xml
DECL INT I_part=10
DECL BOOL B_error=TRUE
DECL REAL R_price=22.2000
ENUM COLOR_TYPE red, green, blue,yellow
DECL COLOR_TYPE E_color=#red
```

```
...
```

WaitForUserList.xml

File C:\KRC\USER\WaitForUserList.xml in default state:

```
...
 <WaitForUserList Monitor="False" OperandLevels="12">
<!--ListParam Item="VariableA" ListMode="Active"/-->
<!--ListParam Item="VariableB" ListMode="Active"/-->
<!--ListParam Item="VariableC" ListMode="Active"/-->
<!--ListParam Item="VariableD" ListMode="Active"/-->
 </WaitForUserList>
...
```

Adapt file WaitForUserList.xml:

- Enter the variable names.
- If the predefined example lines are used, these must be commented out.

```
...
 <WaitForUserList Monitor="False" OperandLevels="12">
<ListParam Item="I_part" ListMode="Active"/>
<ListParam Item="B_error" ListMode="Active"/>
<ListParam Item="R_price" ListMode="Active"/>
<ListParam Item="E_color" ListMode="Active"/>
 </WaitForUserList>
...
```

`Monitor` and `ListMode` are not evaluated by the robot controller. If they have different values than shown above, this therefore has no effect.

Program

WAIT FOR inline form with user list:

After the cold start: touching the **User list** button displays a selection box with the user-defined variables in the inline form.

Select a variable from the list:

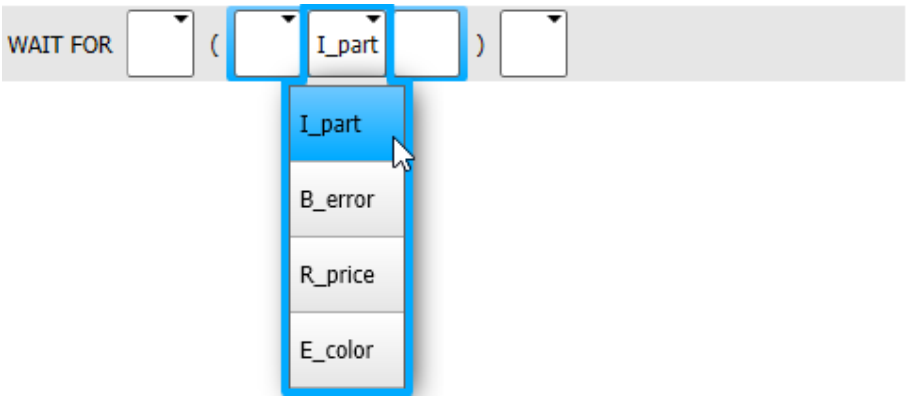


Fig. 6-23: Example: WAIT FOR with user list

Enter the desired value in the box to the right of the variable:

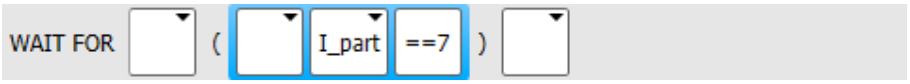


Fig. 6-24: Example: WAIT FOR with variable from user list

6.21 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. 6-25: 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.
MOVEMENTS	Type: INT Maximum number of motions recorded for backward motion • **0 … 60** (default = 30)
BACKWARD_WARNING	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

6.22 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. (>>> *6.22.1 "Configuring CELL.SRC" Page 277*)
2	Configuration of the inputs/outputs of the Automatic External interface. (>>> *6.22.2 "Configuring Automatic External inputs/outputs" Page 278*)
3	Only if error numbers are to be transmitted to the higher-level controller: configuration of the P00.DAT file. (>>> *6.22.6 "Transmitting error numbers to the higher-level controller" Page 294*)

6.22.1 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
...
```

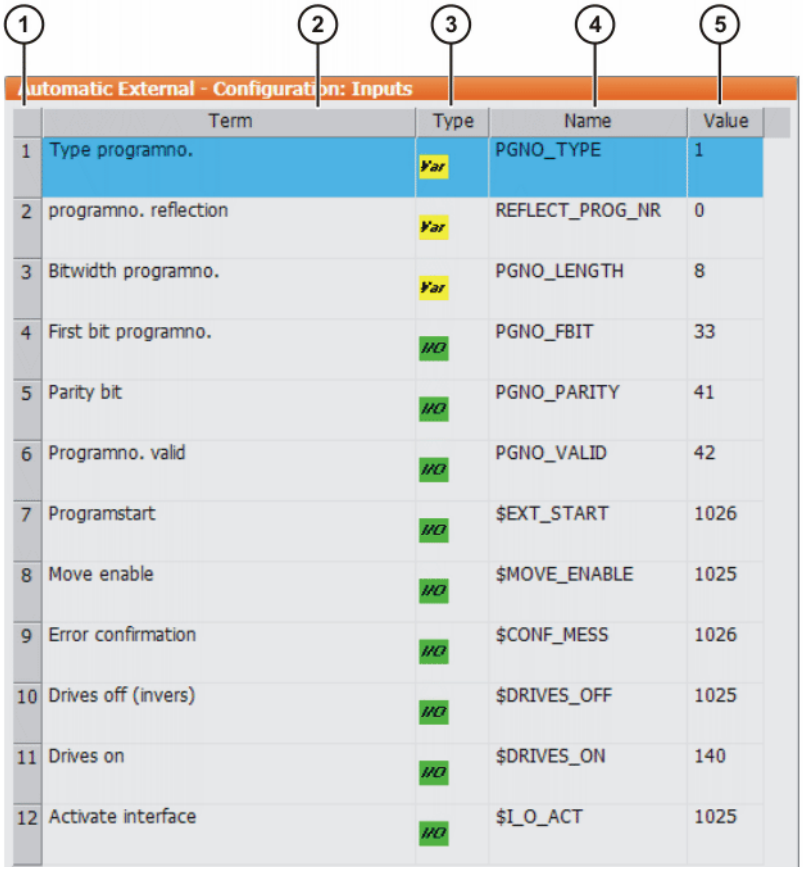
3. For all further programs, proceed as described in step 2. If required, add additional CASE branches.
4. Close the program CELL.SRC. Respond to the request for confirmation asking whether the changes should be saved with **Yes**.

6.22.2 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. 6-26: Configuring Automatic External inputs



Fig. 6-27: 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.

6.22.3 Automatic External: inputs

6.22.3.1 PGNO_TYPE

Description

Variable for configuring the “Automatic External” interface

This variable defines the format in which the program number sent by the higher-level controller is read.

Writability

The variable can be modified in the following way:

- Via the variable display
- Or via **Configuration** > **Inputs/outputs** > **Automatic External**

Syntax

```
PGNO_TYPE = Format
```

Explanation of the syntax

Element	Description
Format	Type: INT • **1** (default): binary number The program number is transmitted by the higher-level controller as a binary coded integer. • **2**: BCD value The program number is transmitted by the higher-level controller as a binary coded decimal. • **3**: “1 of N” The program number is transmitted by the higher-level controller or the periphery as a “1 of n” coded value. **Note:** When using this format, the values of PGNO_REQ, PGNO_PARITY and PGNO_VALID are not evaluated and are thus of no significance.

Examples

Example 1:

```
PGNO_TYPE = 1
```

0 0 1 0 0 1 1 1 => Program number = 39

Example 2:

```
PGNO_TYPE = 2
```

0 0 1 0 0 1 1 1 => Program number = 27

Example 3:

```
PGNO_TYPE = 3
```

0 0 0 0 0 0 0 1 => Program number = 1

0 0 0 0 1 0 0 0 => Program number = 4

6.22.3.2 REFLECT_PROG_NR

Description

Variable for configuring the “Automatic External” interface

The variable defines whether the program number that the robot controller receives from the higher-level controller is mirrored to a range of outputs. The range starts with the output defined by PGNO_FBIT_REFL.

Writability

The variable can be modified in the following way:

- Via the variable display
- Or via **Configuration** > **Inputs/outputs** > **Automatic External**

Syntax

```
REFLECT_PROG_NR = Status
```

Explanation of the syntax

Element	Description
Status	Type: INT • **0** (default): mirroring is switched off. • **1**: mirroring is switched on.

6.22.3.3 PGNO_LENGTH

Description

Variable for configuring the “Automatic External” interface
Number of bits in the program number sent by the higher-level controller

Writability

The variable can be modified in the following way:

- Via the variable display
- Or via **Configuration** > **Inputs/outputs** > **Automatic External**

Syntax

`PGNO_LENGTH =` *Length*

Explanation of the syntax

Element	Description
Length	Type: INT • **1 … 16**: the number of bits in the program number sent by the higher-level controller Default: 8 **Note:** If PGNO_TYPE has the value 2, only 4, 8, 12 and 16 are permissible values for the number of bits.

6.22.3.4 PGNO_FBIT

Description

Variable for configuring the “Automatic External” interface
The variable specifies the input that receives the first bit of the program number transferred by the higher-level controller.

Writability

The variable can be modified in the following way:

- Via the variable display
- Or via **Configuration** > **Inputs/outputs** > **Automatic External**

Syntax

`PGNO_FBIT =` *Input_number*

Explanation of the syntax

Element	Description
Input_number	Type: INT • **1 … 8192**: input that receives the first bit of the program number sent by the higher-level controller Default: 33

Example

PGNO_FBIT = 5 => the external program number begins with the input $IN[5].

6.22.3.5 PGNO_PARITY

Description

Variable for configuring the “Automatic External” interface

The variable specifies the input to which the parity bit is sent by the higher-level controller.

Writability

The variable can be modified in the following way:

- Via the variable display
- Or via **Configuration** > **Inputs/outputs** > **Automatic External**

Syntax

`PGNO_PARITY =` *Input_number*

Explanation of the syntax

Element	Description
Input_number	Type: INT • **1 … 8192**: input to which the parity bit is sent by the higher-level controller Default: 41 **Note:** If PGNO_TYPE has the value 3, PGNO_PARITY is not evaluated.

Input	Function
Negative value	Odd parity
0	No evaluation
Positive value	Even parity

(>>> *6.22.4 "Odd / even parity" Page 286*)

6.22.3.6 PGNO_VALID

Description

Variable for configuring the “Automatic External” interface

The variable specifies the input to which the command for reading the program number from the higher-level controller is sent.

Writability

The variable can be modified in the following way:

- Via the variable display
- Or via **Configuration** > **Inputs/outputs** > **Automatic External**

Syntax

```
PGNO_VALID = Input_number
```

Explanation of the syntax

Element	Description
Input_number	Type: INT • **1 … 8192**: input to which the command to read the program number from the higher-level controller is sent Default: 42 **Note:** If PGNO_TYPE has the value 3, PGNO_VALID is not evaluated.

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.

6.22.3.7 $EXT_START

Description

Signal declaration for the external program start

If the Automatic External interface is active ($I_O_ACTCONF is **TRUE**), this input can be set by the higher-level controller to start or continue a program.

	Only the rising edge of the signal is evaluated.

Syntax

```
SIGNAL $EXT_START $IN[Input number]
```

Explanation of the syntax

Element	Description
Input number	Type: INT Default: **1026**

6.22.3.8 $MOVE_ENABLE

Description

Signal declaration for motion enable

This input is used by the higher-level controller to check the robot drives.

- If the higher-level controller sets the input to **TRUE**, the robot can be moved manually and in program mode.
- If the input is set to **FALSE**, the drives are switched off and the active commands are 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 acknowledged and an external start signal ($EXT_START) has been given.

Syntax

```
SIGNAL $MOVE_ENABLE $IN[Input number]
```

Explanation of the syntax

Element	Description
Input number	Type: INT Default: **1025**

6.22.3.9 $CHCK_MOVENA

Description

Check the input number of $MOVE_ENABLE in Automatic External mode

The variable $MOVE_ENABLE is used by the higher-level controller to check the robot drives. A precondition is that a suitable input number (<> $IN[1025]) has been assigned to the signal $MOVE_ENABLE. This is checked with $CHCK_MOVENA.

Background: by default, $IN[1025] is a system input which is always **TRUE**. Consequently, this input must not be allocated to the system variable $MOVE_ENABLE as this would mean that the motion enable would be issued on a permanent basis.

Syntax

```
$CHCK_MOVENA=State
```

Explanation of the syntax

Element	Description
Status	Type: BOOL • TRUE: $MOVE_ENABLE is checked. • FALSE: $MOVE_ENABLE is not checked. Default: TRUE

6.22.3.10 $CONF_MESS

Description

Signal declaration for the external acknowledgement of messages

If the Automatic External interface is active ($I_O_ACT is TRUE), this input can be set by the higher-level controller to acknowledge an error message as soon as the cause of the error has been eliminated.

Only the rising edge of the signal is evaluated.

Syntax

```
SIGNAL $CONF_MESS $IN[Input number]
```

Explanation of the syntax

Element	Description
Input number	Type: INT Default: **1026**

6.22.3.11 $DRIVES_OFF

Description

Signal declaration for switching off the drives

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.

Syntax

```
SIGNAL $DRIVES_OFF $IN[Input number]
```

Explanation of the syntax

Element	Description
Input number	Type: INT Default: **1025**

6.22.3.12 $DRIVES_ON

Description

Signal declaration for switching on the drives

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.

Syntax

```
SIGNAL $DRIVES_ON $IN[Input number]
```

Explanation of the syntax

Element	Description
Input number	Type: INT Default: **140**

6.22.3.13 $I_O_ACT

Description

Signal declaration for the Automatic External interface

If this input is set, the Automatic External interface is active.

Syntax

```
SIGNAL $I_O_ACT $IN[Input number]
```

Explanation of the syntax

Element	Description
Input number	Type: INT Default: **1025**

6.22.4 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

6.22.5 Automatic External: outputs

6.22.5.1 $RC_RDY1

Description

Signal declaration for operational robot controller

If the robot controller sets this output, it is ready for operation and the program can be started by the higher-level controller.

Syntax

```
SIGNAL $RC_RDY1 $OUT[Output number]
```

Explanation of the syntax

Element	Description
Output number	Type: INT Default: **137**

6.22.5.2 $ALARM_STOP

Description

Signal declaration for a digital output

- **FALSE**: The safety controller has triggered a safety STOP 0 or safety STOP 1, e.g. due to an EMERGENCY STOP.
- **TRUE**: The safety controller has possibly triggered a safety stop.
  No EMERGENCY STOP is active.

The output is set to FALSE in the following EMERGENCY STOP situations:

- EMERGENCY STOP via the EMERGENCY STOP device on the smartPAD
- EMERGENCY STOP via an external EMERGENCY STOP device
- Only for KR C5: EMERGENCY STOP via the EMERGENCY STOP device on the cabinet door (if present)



$ALARM_STOP_INTERN indicates whether an EMERGENCY STOP has been triggered internally or externally.
(>>> *6.22.5.9 "$ALARM_STOP_INTERN" Page 290*)

Syntax

```
SIGNAL $ALARM_STOP $OUT[Output number]
```

Explanation of the syntax

Element	Description
Output number	Type: INT Default: **1013**

6.22.5.3 $USER_SAF

Description

Signal declaration for the safety fence monitoring

This output is reset if the safety fence monitoring switch is opened (AUT mode) or an enabling switch is released (T1 or T2 mode).

Syntax

```
SIGNAL $USER_SAF $OUT[Output number]
```

Explanation of the syntax

Element	Description
Output number	Type: INT Default: **1011**

6.22.5.4 $PERI_RDY

Description

Signal declaration for Drives ON

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.

Syntax

```
SIGNAL $PERI_RDY $OUT[Output number]
```

Explanation of the syntax

Element	Description
Output number	Type: INT Default: **1012**

6.22.5.5 $ROB_CAL

Description

Signal declaration for robot mastering

This output is set whenever all robot axes are mastered. The output is reset as soon as a robot axis has been unmastered.

Syntax

```
SIGNAL $ROB_CAL $OUT[Output number]
```

Explanation of the syntax

Element	Description
Output number	Type: INT Default: **1001**

6.22.5.6 $I_O_ACTCONF

Description

Signal declaration for the active Automatic External interface

This output is set if Automatic External mode is selected and the Automatic External interface is active (input $I_O_ACT is TRUE). The higher-level controller can start a program.

Syntax

```
SIGNAL $I_O_ACTCONF $OUT[Output number]
```

Explanation of the syntax

Element	Description
Output number	Type: INT Default: **140**

6.22.5.7 $STOPMESS

Description

Signal declaration for stop messages

This output is set in order to communicate to the higher-level controller the occurrence of any message that required the robot to be stopped. For example, after an EMERGENCY STOP or operator safety violation.

Syntax

```
SIGNAL $STOPMESS $OUT[Output number]
```

Explanation of the syntax

Element	Description
Output number	Type: INT Default: **1010**

6.22.5.8 PGNO_FBIT_REFL

Description

Variable for configuring the “Automatic External” interface

The variable specifies the output to which the first bit of the program number is mirrored. A precondition for the program number to be mirrored at all is REFLECT_PROG_NR = 1.

The size of the output range 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 range 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 located in the CELL program.

Writability

The variable can be modified in the following way:

- Via the variable display
- Or via **Configuration** > **Inputs/outputs** > **Automatic External**

Syntax

```
PGNO_FBIT_REFL = Output_number
```

Explanation of the syntax

Element	Description
Output_number	Type: INT • **1 … 8192**: output representing the first bit of the program number transferred by the higher-level controller Default: 999

6.22.5.9 $ALARM_STOP_INTERN

Description

Signal declaration for the internal EMERGENCY STOP

- **FALSE**: An internal EMERGENCY STOP is active.
- **TRUE**: No EMERGENCY STOP or external EMERGENCY STOP is active.

Internal EMERGENCY STOP means:

- EMERGENCY STOP via the EMERGENCY STOP device on the smartPAD
- Only for KR C5: EMERGENCY STOP via the EMERGENCY STOP device on the cabinet door (if present)

Syntax

```
SIGNAL $ALARM_STOP_INTERN $OUT[Output number]
```

Explanation of the syntax

Element	Description
Output number	Type: INT Default: **1002**

6.22.5.10 $PRO_ACT

Description

Signal declaration for active process

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).

Syntax

```
SIGNAL $PRO_ACT $OUT[Output number]
```

Explanation of the syntax

Element	Description
Output number	Type: INT Default: **1021**

6.22.5.11 PGNO_REQ

Description

Variable for configuring the “Automatic External” interface

The variable specifies an output. A change of signal at this output requests the higher-level controller to send a program number.

Writability

The variable can be modified in the following way:

- Via the variable display
- Or via **Configuration** > **Inputs/outputs** > **Automatic External**

Syntax

```
PGNO_REQ = Output_number
```

Explanation of the syntax

Element	Description
Output_number	Type: INT • **1 … 8192**: a change of signal at this output requests the higher-level controller to send a program number. Default: 33 **Note:** If PGNO_TYPE has the value 3, PGNO_REQ is not evaluated.

6.22.5.12 APPL_RUN

Description

Variable for configuring the “Automatic External” interface

The variable specifies an output. By setting this output, the robot controller communicates to the higher-level controller the fact that a program is currently being executed.

Writability

The variable can be modified in the following way:

- Via the variable display
- Or via **Configuration** > **Inputs/outputs** > **Automatic External**

Syntax

```
APPL_RUN = Output_number
```

Explanation of the syntax

Element	Description
Output_number	Type: INT • **1 … 8192**: by setting this output, the robot controller communicates to the higher-level controller the fact that a program is currently being executed. Default: 34

6.22.5.13 $PRO_MOVE

Description

Signal declaration for active program motion

This output is set whenever a synchronous axis moves (also in jog mode). The signal is thus the inverse of $ROB_STOPPED.

Syntax

```
SIGNAL $PRO_MOVE $OUT[Output number]
```

Explanation of the syntax

Element	Description
Output number	Type: INT Default: **1022**

6.22.5.14 $IN_HOME

Description

Signal declaration for reaching the HOME position

The variable $H_POS defines the HOME position of the robot in the machine data. By setting the output, the robot controller communicates to the higher-level controller that the robot is located in its HOME position.

Syntax

```
SIGNAL $IN_HOME $OUT[Output number]
```

Explanation of the syntax

Element	Description
Output number	Type: INT Default: **1000**

6.22.5.15 $IN_HOME1 … $IN_HOME5

Description

Signal declaration for reaching HOME position 1 ... 5

The variable $AXIS_HOME allows up to 5 HOME positions to be defined in $machine.dat under KRC:\R1\Mada (in addition to the HOME position defined using $H_POS).

By setting the output $IN_HOME*x*, the robot controller communicates to the higher-level controller that the robot is located in HOME position *x* (1 ... 5).

Syntax

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

`SIGNAL $IN_HOME`*index* `FALSE`

To link an output to an $IN_HOME, make the following adaptation:

`SIGNAL $IN_HOME`*Index* `$OUT[`*Output number*`]`

Explanation of the syntax

Element	Description
Index	Type: INT Index of the HOME position • **1 … 5**
Output number	Type: INT

6.22.5.16 $ON_PATH

Description

Signal declaration for monitoring of the programmed path

This output is set after the BCO run. The robot controller thus communicates to the higher-level controller that the robot is located on the programmed path. The output is reset again only if the robot leaves the path, the program is reset or block selection is carried out.

Syntax

`SIGNAL $ON_PATH $OUT[`*Output number*`]`

Explanation of the syntax

Element	Description
Output number	Type: INT Default: **997**

6.22.5.17 $NEAR_POSRET

Description

Signal declaration for the tolerance window about $POS_RET

By setting the output, the robot controller communicates to the higher-level controller that the robot is located 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 variable $NEARPATHTOL.

Syntax

```
SIGNAL $NEAR_POSRET $OUT[Output number]
```

Explanation of the syntax

Element	Description
Output number	Type: INT Default: **147**

6.22.5.18 $ROB_STOPPED

Description

Signal declaration for robot standstill

This output 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.

Syntax

```
SIGNAL $ROB_STOPPED $OUT[Output number]
```

Explanation of the syntax

Element	Description
Output number	Type: INT Default: **1023**

6.22.6 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.

6.22.7 Signal diagrams

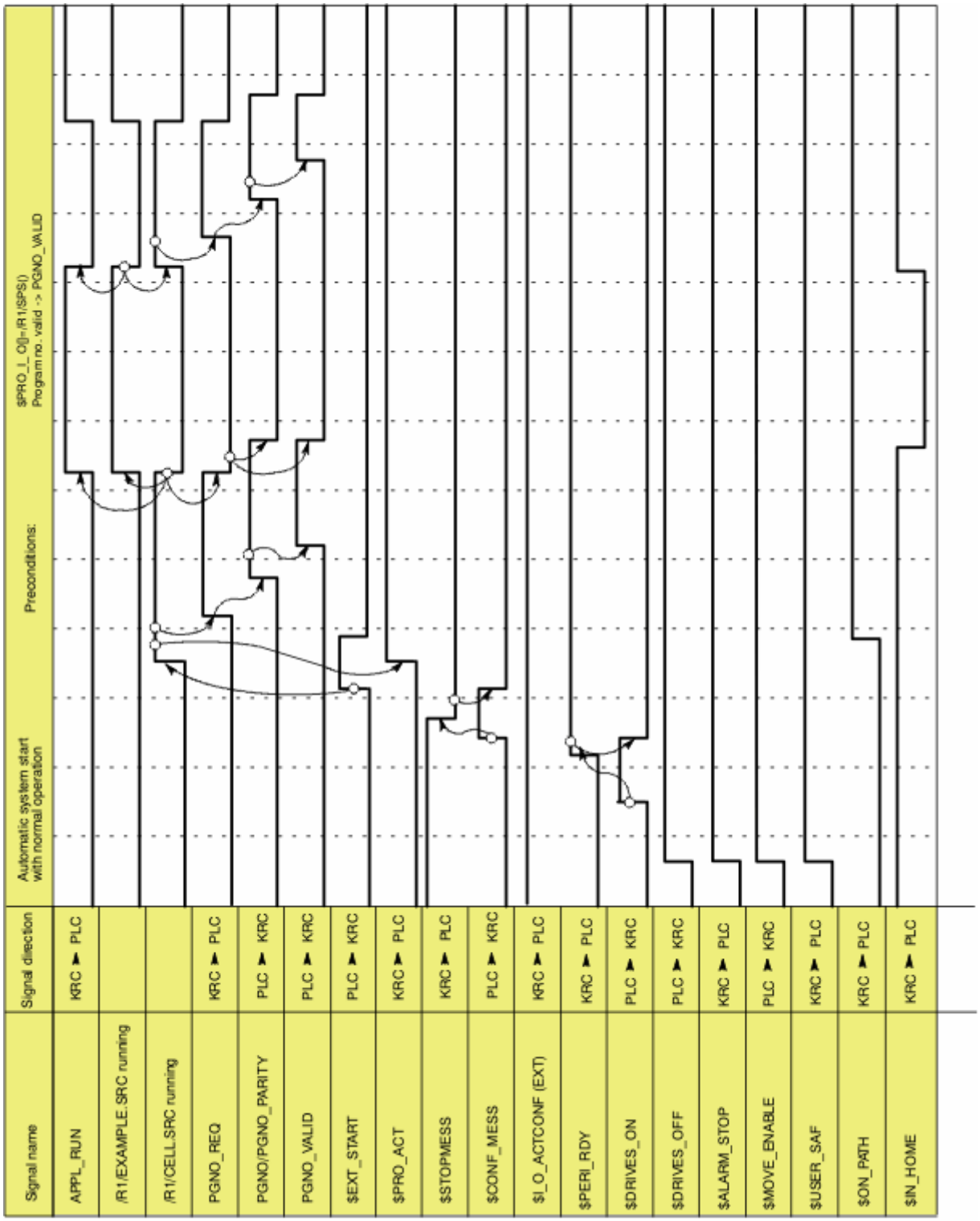


Fig. 6-28: Automatic system start and normal operation with program number acknowledgement by means of PGNO_VALID

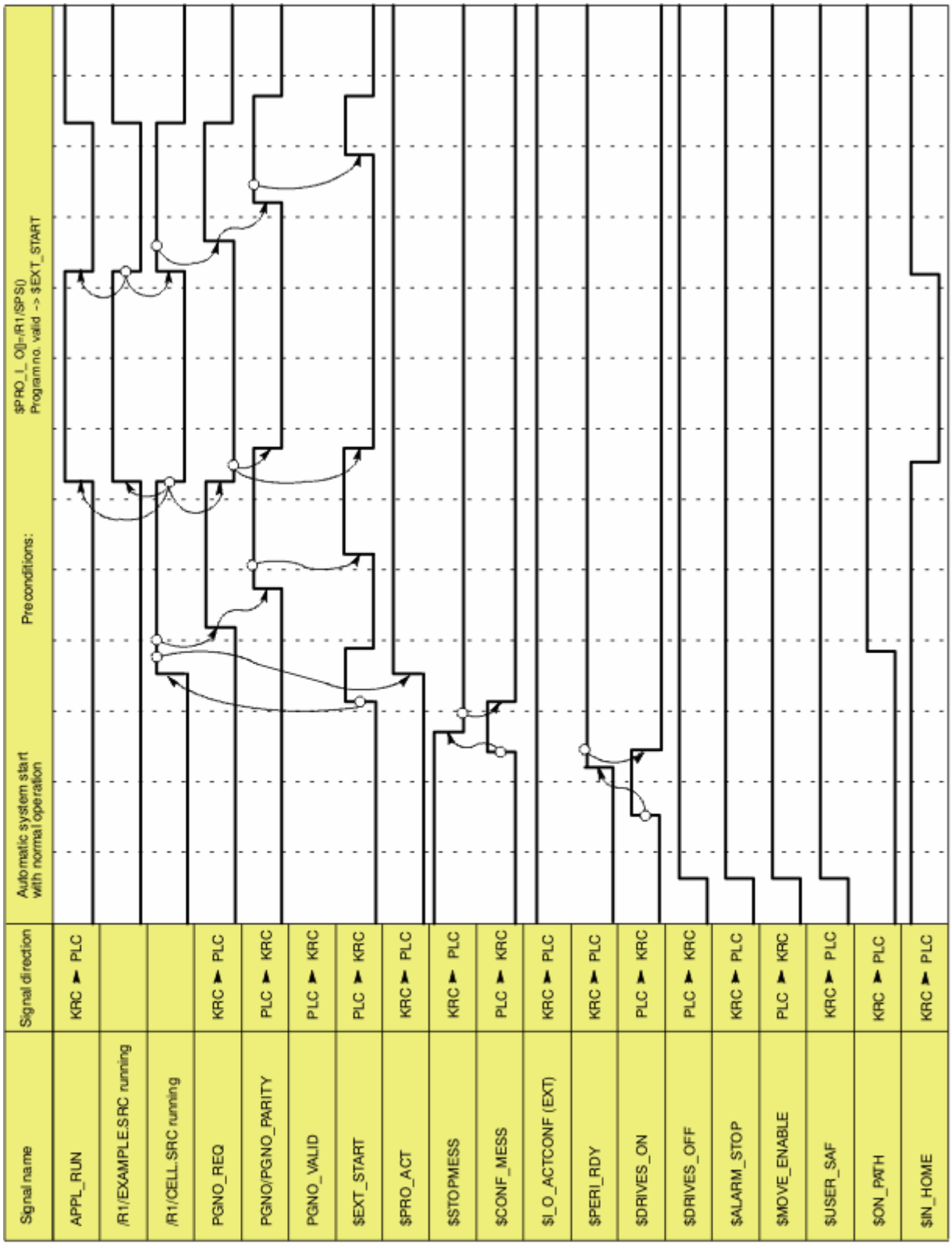


Fig. 6-29: Automatic system start and normal operation with program number acknowledgement by means of $EXT_START

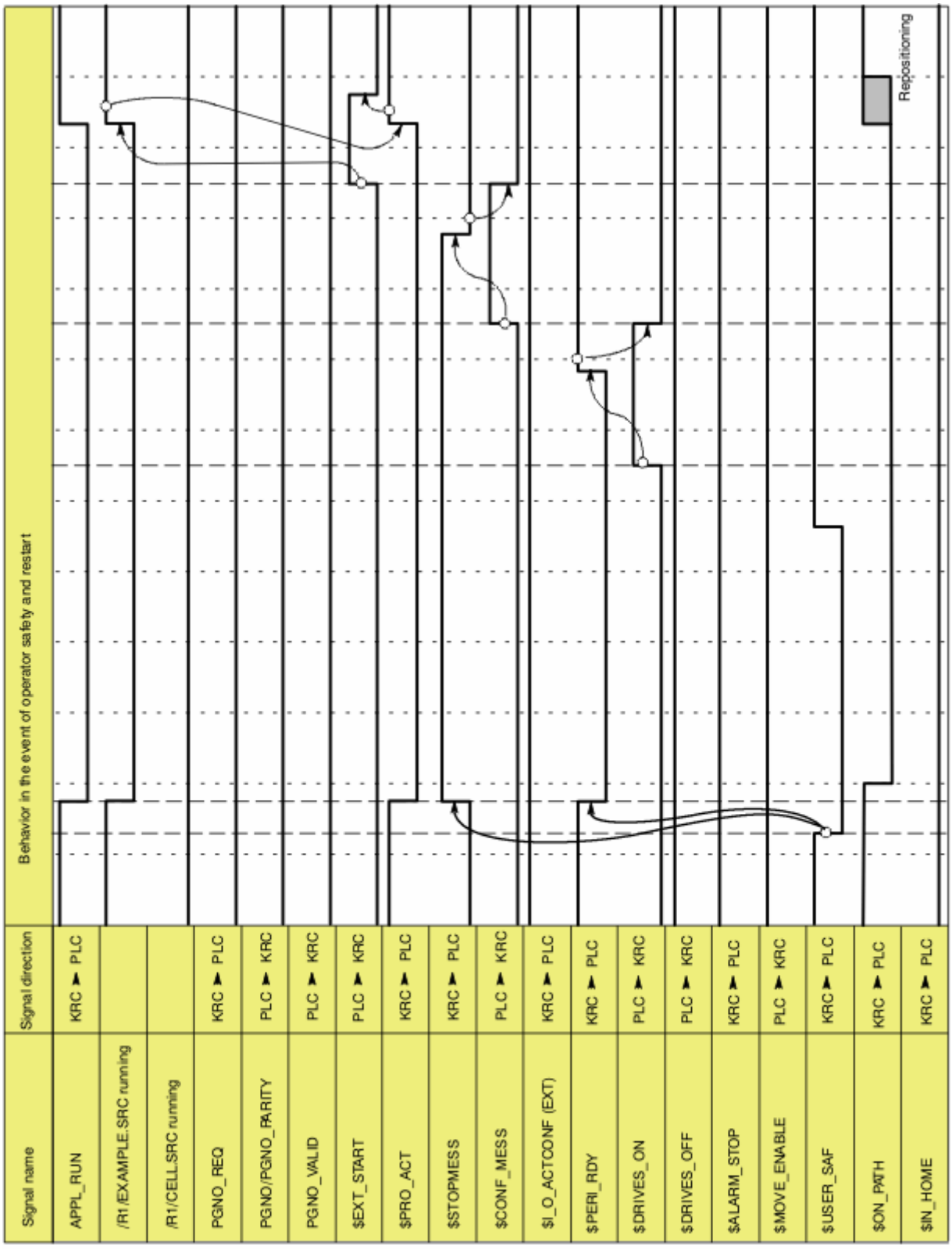


Fig. 6-30: Restart after dynamic braking (operator safety and restart)

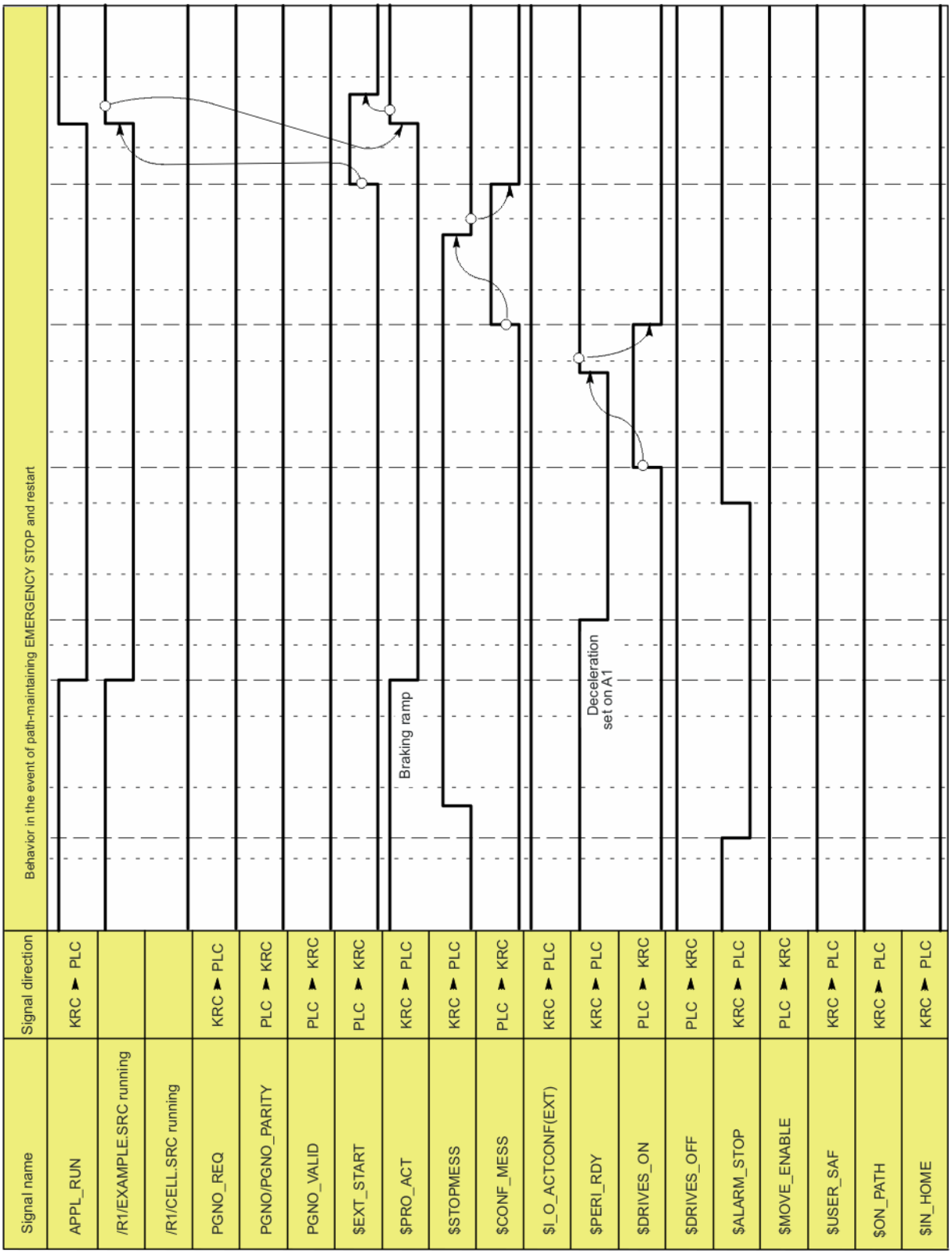


Fig. 6-31: Restart after path-maintaining EMERGENCY STOP

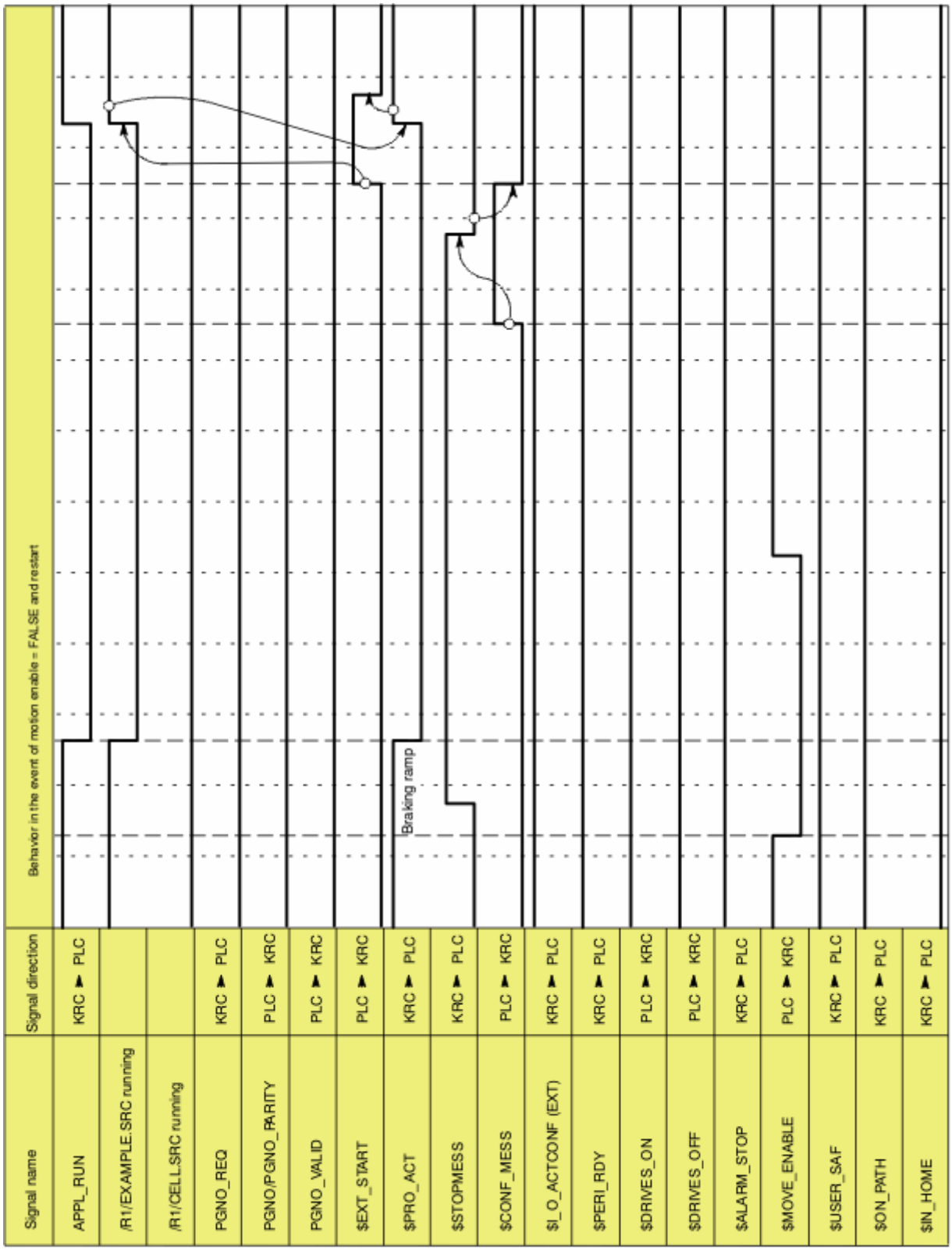


Fig. 6-32: Restart after motion enable

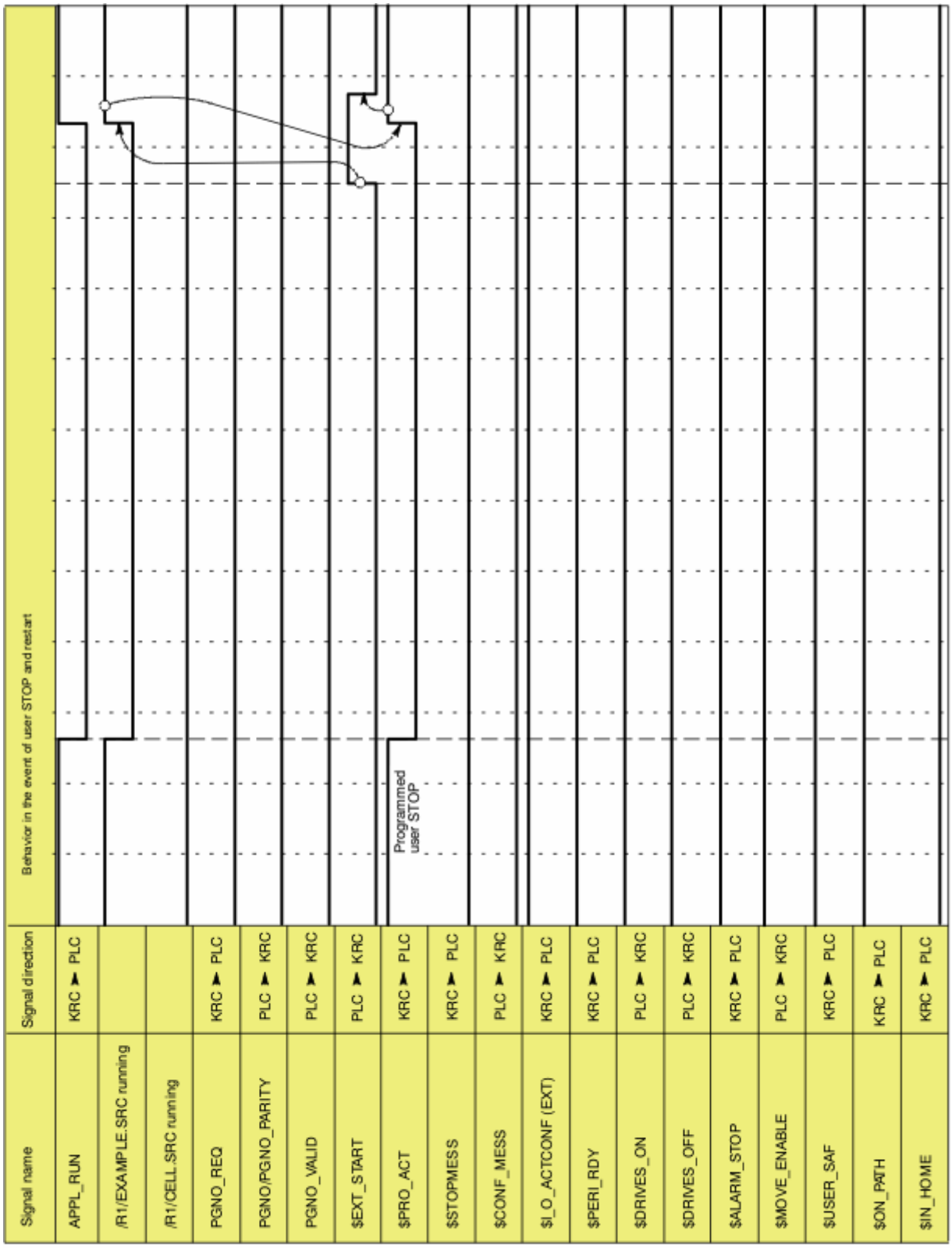


Fig. 6-33: Restart after user STOP

6.23 Torque mode

6.23.1 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.

6.23.1.1 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.
   (>>> *6.23.2 "SET_TORQUE_LIMITS: activating torque mode" Page 305*)
   If the regular monitoring functions are deactivated, other monitoring functions specially adapted to torque mode are automatically activated.
   (>>> *"Monitoring functions" Page 304*)
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.
   (>>> *6.23.3 "RESET_TORQUE_LIMITS: deactivating torque mode" Page 308*)
   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 $TORQUE_AXIS_MAX interval.
  (>>> *6.23.5.3 "$TORQUE_AXIS_MAX" Page 311*)
- 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 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.

(>>> *6.23.6.2 "Torque mode program example: avoiding damage in the event of collisions" Page 314*)

6.23.1.2 Torque mode 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.

(>>> *6.23.6.1 "Torque mode program example: setting axis to “soft” in both directions" Page 313*)

6.23.2 SET_TORQUE_LIMITS: activating torque mode

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())
```

6.23.3 RESET_TORQUE_LIMITS: deactivating torque mode

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 (>>> *11.8.1 "CONTINUE: preventing an advance run stop" Page 594*)!

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.

6.23.4 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.)

  (>>> *6.23.6.3 "Program example: torque mode in the interrupt" Page 315*)

- 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.

6.23.5 Diagnostic variables for torque mode

All these variables and constants are write-protected.

Their value is not dependent on the interpreter.

6.23.5.1 $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.

(>>> *6.23.5.6 "Comparison: $TORQUE_AXIS_ACT and $HOLDING_TORQUE " Page 313*)

$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.

6.23.5.2 $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.

6.23.5.3 $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.

6.23.5.4 $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 display or variable overview.

Properties:

- 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 system variable $TORQUE_AXIS_LIMITS is accessed via KRL, however, the robot controller may regard the access as “invalid”. Recommendation: Check the state of the system variable with VARSTATE() prior to access.

Advance run stop: in the robot program, the variable triggers an advance run stop.

6.23.5.5 $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.

(>>> *6.23.5.6 "Comparison: $TORQUE_AXIS_ACT and $HOLDING_TORQUE " Page 313*)

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.

6.23.5.6 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.

6.23.6 Other examples

6.23.6.1 Torque mode program example: 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

```
 1 DECL TorqLimitParam myLimits
 2 DECL INT ideal_axis
   ...
 3 myLimits.monitor = #off
 4 myLimits.lower = $holding_torque[ideal_axis] - 10
 5 myLimits.upper = $holding_torque[ideal_axis] + 10
 6 SET_TORQUE_LIMITS(ideal_axis, myLimits)
 7 PTP $AXIS_ACT
 8 OUT_SIGNAL_SOFT = TRUE
 9 WAIT FOR IN_SIGNAL_EJECTED
10 RESET_TORQUE_LIMITS(ideal_axis)
11 OUT_SIGNAL_SOFT = FALSE
12 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.

6.23.6.2 Torque mode program example: 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

6.23.6.3 Program example: 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 to 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.

6.23.6.4 Torque mode program example: servo gun builds up pressure (robot program)

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_ACTIVE]
 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 to 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.

6.23.6.5 Torque mode program example: servo gun builds up pressure (submit program)

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.

6.24 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**.

6.25 Brake test

6.25.1 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.

Operating modes

The brake test can be performed in all operating modes.

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.

6.25.2 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.

6.25.3 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. (>>> *6.25.5.1 "Performing a brake test for requested axes (cyclically via program)" Page 330*)
	BrakeTestReq.src can also be selected manually. All active axes are tested. (>>> *6.25.5.2 "Performing a brake test for active axes (manually)" Page 332*)
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. (>>> *6.25.5.3 "Performing a brake test for further axes (e.g. couplable axes)" Page 334*)
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.

6.25.4 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: (>>> *6.25.4.1 "Activating the brake test, defining the cycle time and axes" Page 322*) • 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. (>>> *6.25.4.3 "Configuring input and output signals for the brake test" Page 324*)
3	Teach positions for the brake test. (>>> *6.25.4.5 "Teaching positions for the brake test" Page 328*)
4	Test the sequence. (>>> *6.25.4.6 "Testing the sequence in the case of defective brakes" Page 329*)

6.25.4.1 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.
   (>>> *6.25.4.2 "“Brake test configuration” window" Page 323*)
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.

6.25.4.2 “Brake test configuration” window



Fig. 6-34: “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.

6.25.4.3 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

Danger to life and limb due to incorrectly used signals
These signals are not redundant in design and can supply incorrect information. If they are used for safety-relevant applications, this may result in death, severe injuries or considerable damage to property.

- 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.)

Signal	Description
$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).
$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*

6.25.4.4 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.

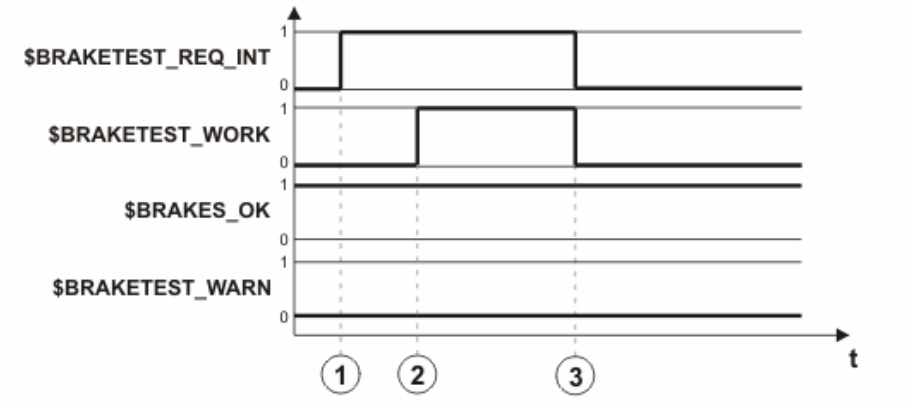


Fig. 6-35: Signal diagram: brakes OK

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.

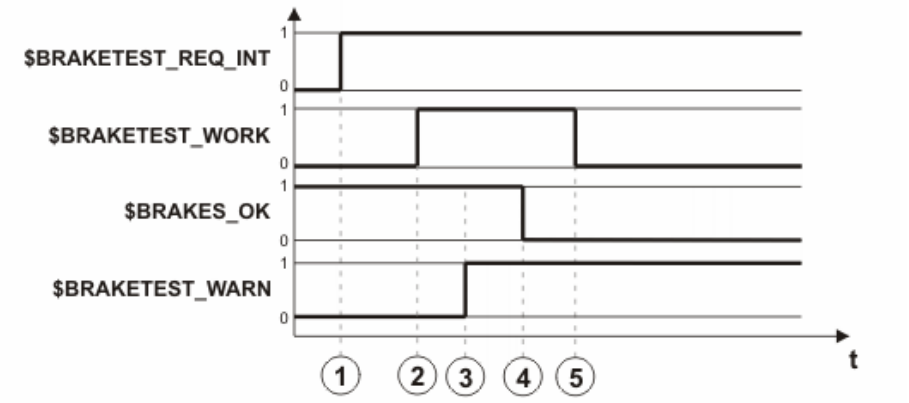


Fig. 6-36: Signal diagram: brakes not OK

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.
5	The robot has been moved to the parking position or the program has been canceled.

6.25.4.5 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

A suitable 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

Danger to life and limb due to unfavorable parking position
A defective brake may cause the robot to sag, even if it is already in the parking position.

- Select the parking position where no persons are endangered if the robot sags.

  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.

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.
(>>> *6.25.4.6 "Testing the sequence in the case of defective brakes" Page 329*)

6.25.4.6 Testing the sequence in the case of defective brakes

WARNING

Danger to life and limb due to unfavorable parking position
A defective brake may cause the robot to sag, even if it is already in the parking position.

- Select the parking position where no persons are endangered if the robot sags.

  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.

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.

6.25.5 Performing a brake test

6.25.5.1 Performing a brake test for requested axes (cyclically via program)

WARNING

Danger to life and limb due to entering the danger zone prematurely
A defective brake may cause the robot to sag.
- Do not enter the danger zone until the robot has reached its parking position.

NOTICE

Damage to property during motion to the parking position
If a brake is defective, the robot may sag when it is stopped.
- Avoid stops during the motion to the parking position, as sagging could result in damage to property.
  In particular, it must be taken into consideration that certain actions trigger stops (e.g. E-STOP, opening safety gate, changing operating mode), as do certain monitoring functions (e.g. monitoring spaces).

A defective brake can result in longer stopping distances.
- Approach the parking position at low velocity.

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`.

6.25.5.2 Performing a brake test for active axes (manually)



WARNING

Danger to life and limb due to entering the danger zone prematurely
A defective brake may cause the robot to sag.

- Do not enter the danger zone until the robot has reached its parking position.

NOTICE

Damage to property during motion to the parking position
If a brake is defective, the robot may sag when it is stopped.

- Avoid stops during the motion to the parking position, as sagging could result in damage to property.

  In particular, it must be taken into consideration that certain actions trigger stops (e.g. E-STOP, opening safety gate, changing operating mode), as do certain monitoring functions (e.g. monitoring spaces).

A defective brake can result in longer stopping distances.

- Approach the parking position at low velocity.

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.

6.25.5.3 Performing a brake test for further axes (e.g. couplable axes)

WARNING

Danger to life and limb due to entering the danger zone prematurely
A defective brake may cause the robot to sag.

- Do not enter the danger zone until the robot has reached its parking position.

NOTICE

Damage to property during motion to the parking position
If a brake is defective, the robot may sag when it is stopped.

- Avoid stops during the motion to the parking position, as sagging could result in damage to property.

  In particular, it must be taken into consideration that certain actions trigger stops (e.g. E-STOP, opening safety gate, changing operating mode), as do certain monitoring functions (e.g. monitoring spaces).

A defective brake can result in longer stopping distances.

- Approach the parking position at low velocity.

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.

6.25.6 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.

6.25.7 System functions for the brake test

6.25.7.1 GET_AXESMASK: requesting information about the axes for the brake test

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)
```

6.25.7.2 GET_BRAKETEST_TIME: requesting time values for brake test

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. • #BT_REMAINING_CYCLE_TIME Remaining cycle time • #BT_REMAINING_MON_TIME Remaining monitoring time

Total monitoring time: following the message *Brake test required*, the robot can be moved for another 2 hours.

7 Program and project management

7.1 Creating a new program

Precondition

- The Navigator is displayed.
- User rights: Function group **File operations**
  - Depending on the folders that are to be displayed, the “Expert” user group or higher may be required.

Procedure

1. In the left-hand area of the Navigator, select the folder in which the program is to be created, e.g. the folder **Program**.
2. In the right-hand section of the Navigator, select any element in the folder.
3. Press **New**.
4. Only in the user group “Expert” or higher:
   The **Template selection** window opens. Select the desired template and confirm with **OK**.
5. Enter a name for the program and confirm it with **OK**.

Not all folders allow the creation of programs within them, e.g. the folder **Mada**. In this case, the **New** button remains grayed out.

7.2 Creating a new folder

Precondition

- The Navigator is displayed.
- User rights: Function group **File operations**
  - Depending on the folders that are to be displayed, the “Expert” user group or higher may be required.

Procedure

1. In the left-hand area of the Navigator, select the folder in which the new folder is to be created, e.g. the folder **R1**.
2. Press **New**.
3. Enter a name for the folder and confirm it with **OK**.

Not all folders allow the creation of subfolders within them, e.g. the folder **Mada**. In this case, the **New** button remains grayed out.

7.3 Renaming a file or folder

Description

Only files or folders that have been created by users can be renamed.

Precondition

- The Navigator is displayed.
- User rights: Function group **File operations**

– Depending on the folders that are to be displayed, the “Expert” user group or higher may be required.

Procedure

1. In the right-hand area of the Navigator, select the element that is to be renamed (file or folder).
2. Select **Edit** > **Rename**.
3. Overwrite the name with the new name and confirm with **OK**.

7.4 Navigator file manager

Overview

In the Navigator, the user manages programs and system-specific files.

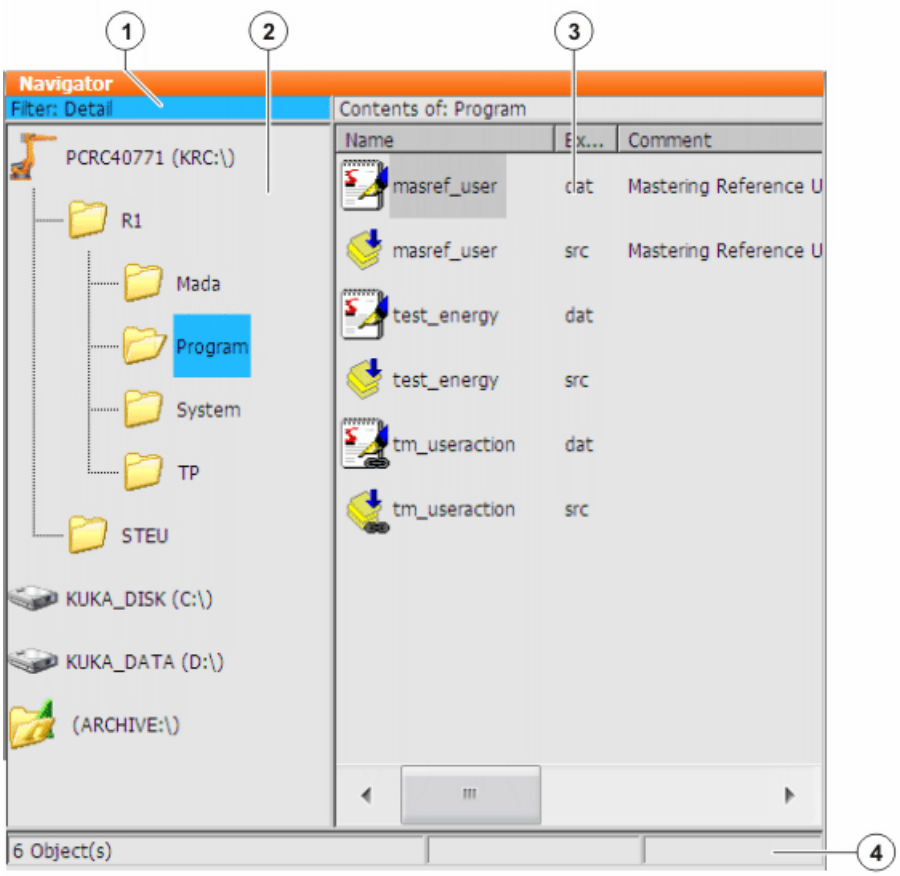


Fig. 7-1: Navigator

1	Header	3	File list
2	Directory structure	4	Status line

Header

- Left: the selected filter is displayed.
  (>>> *7.4.1 "Selecting filters" Page 341*)
- Right: the directory or drive selected in the directory structure is displayed.

Directory structure

Directory structure = left-hand area of the Navigator

Overview of directories and drives. Exactly which directories and drives are displayed depends on the user group and configuration.

File list

File list = right-hand area of the Navigator

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.

Column	Description
Always present:	
Name	Directory or file name
Comment	Comment
Changed	Date and time of the last modification
#	Number of changes made to the file
User group “Expert” or higher:	
Extension	File extension
Attributes	Attributes of the operating system and kernel system
Size	File size in kilobytes
Created	Date and time of file creation

Status bar

The status bar can display the following information:

- Selected objects
- Action in progress
- Requests for confirmation
- User prompts (with a colored background, e.g. *Please enter a name.*)
- Miscellaneous

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

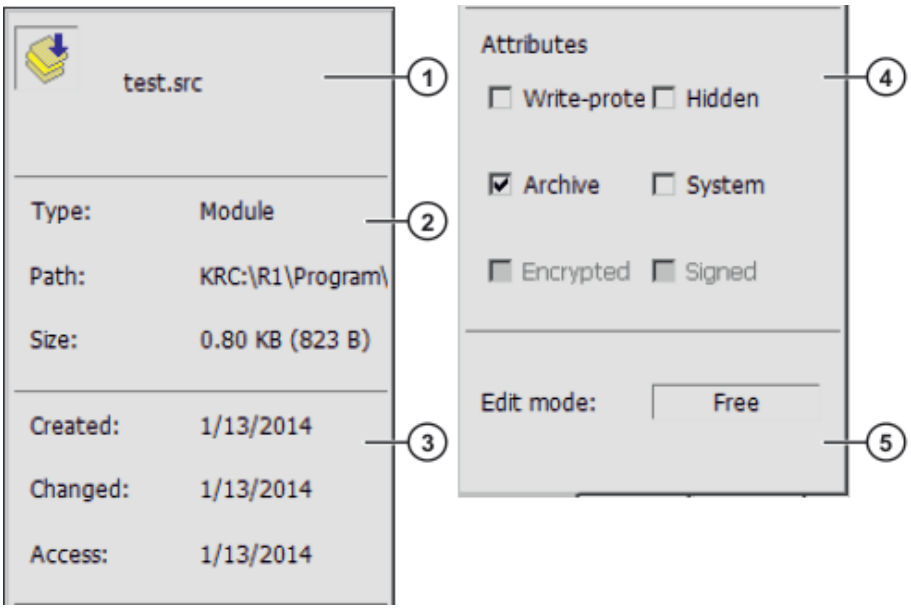


Fig. 7-2: “General” tab

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.

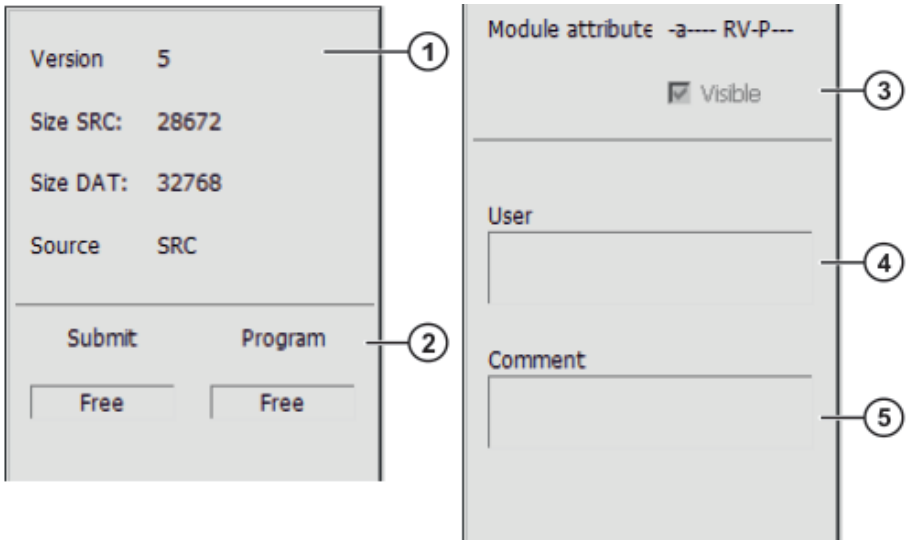


Fig. 7-3: “Module info” tab

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**: A 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. The program is “visible”. • Check box not active: if this program is called as a subprogram, it is not displayed in the Editor. The program is “invisible”. 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.

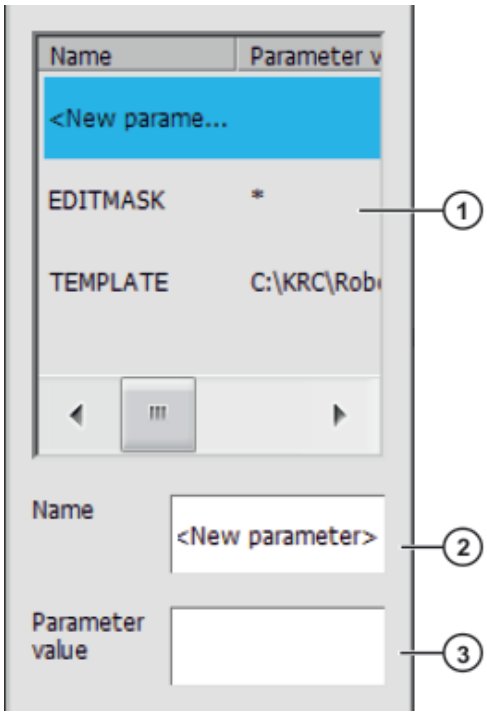


Fig. 7-4: “Parameters” tab

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.

(>>> *8.5 "Robot interpreter status indicator" Page 382*)

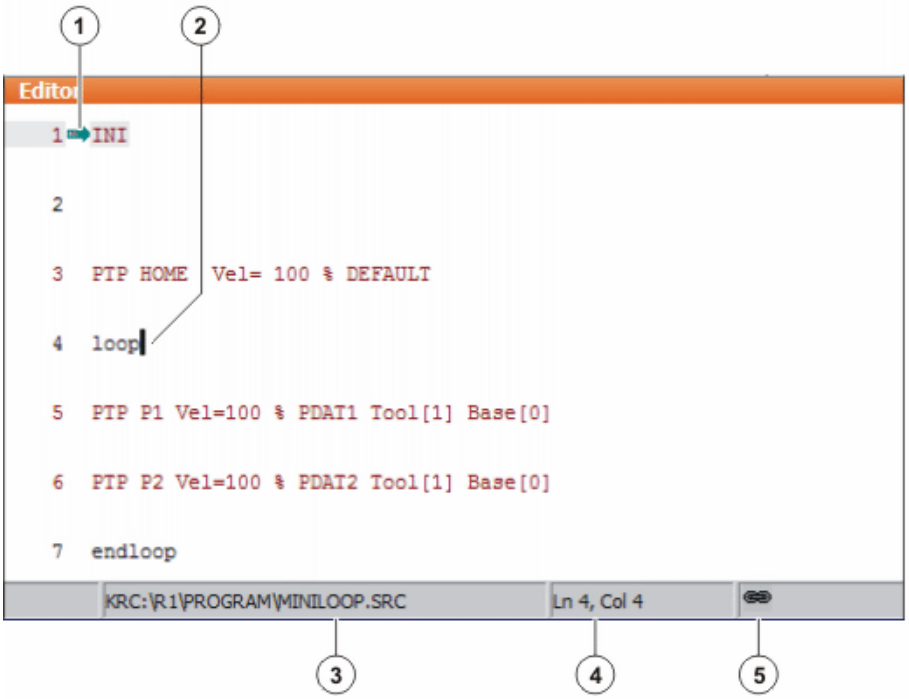


Fig. 7-5: Program is selected.

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

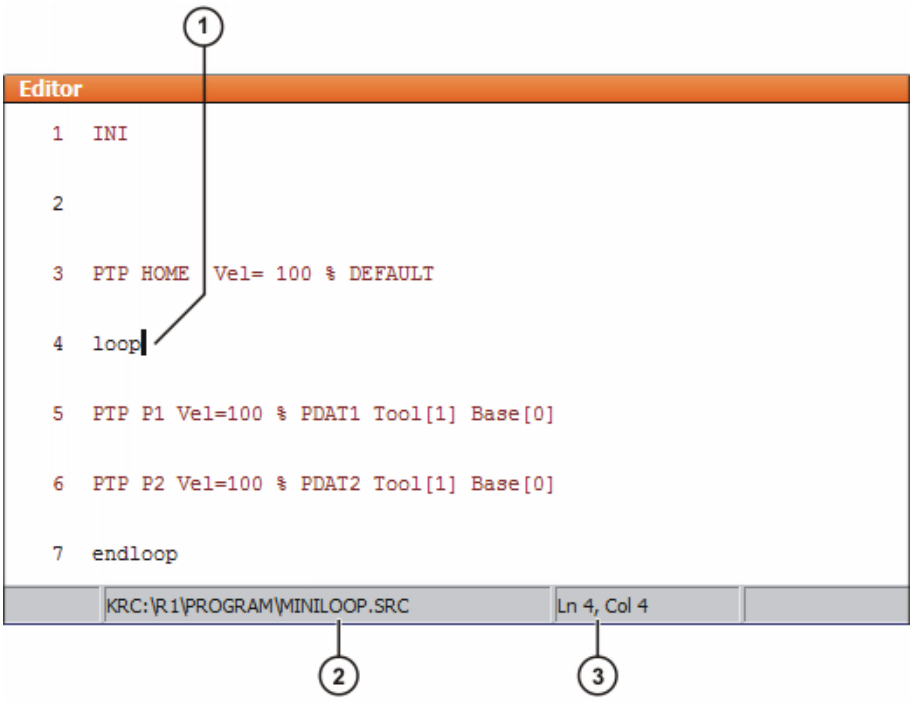


Fig. 7-6: Program is open

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 349*)
2	The INI line contains initializations for internal variables and parameters.
4	HOME position (>>> *7.6.1 "HOME position" Page 348*)
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.

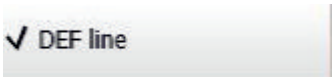
The DEF line is displayed and hidden separately for opened and selected programs. If detail view (ASCII mode) is activated, the DEF line is visible and does not need to be activated separately.

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

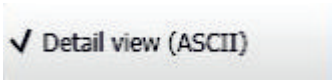
Detail view is deactivated by default to keep programs transparent. If detail view is activated, hidden program lines, such as the FOLD and ENDFOLD lines and the DEF line, are displayed.

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.

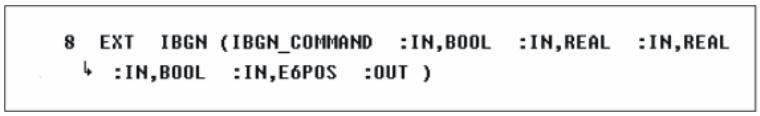


Fig. 7-7: Line break

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.

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 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**.

7.8.2 Selecting a line or range

Precondition

- User rights: Function group **General KRL program changes**
- Program is selected or open.

Procedure

1. Touch the line that is to be selected or in which the selection is to start.
2. Select the menu sequence **Edit** > **Mark region**.
   The line is now selected.
3. To select a range: Touch the line in which the selection is to end.
   The range, including the start and end lines, is now selected.

7.8.3 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. 7-10: 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. 7-11: Creating a sample Fold, step 2

3. Position the cursor in a line outside the Fold. The Fold closes.

```
4
5  outputs
6
```

Fig. 7-12: Creating a sample Fold, step 3

7.8.4 Inserting a comment or stamp in a program

Precondition

- User rights: function group **General KRL program changes**
- T1 mode
- Program is selected or open.

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**.

(>>> *7.8.6 "Display of comments in the program" Page 355*)

Description Comment

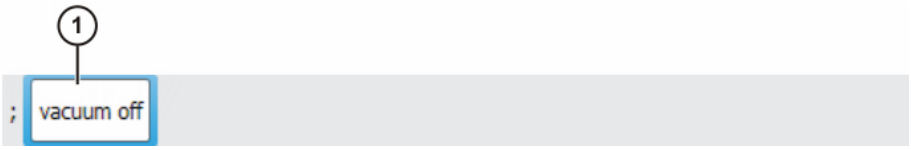


Fig. 7-13: Inline form “Comment”

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.

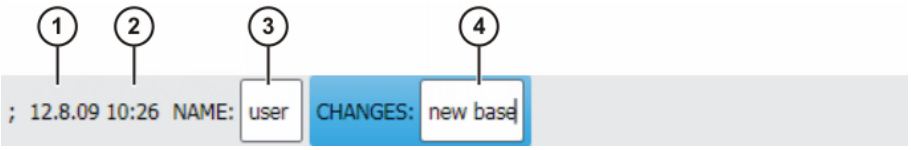


Fig. 7-14: Inline form “Stamp”

Item	Description
1	System date (cannot be edited)
2	System time
3	Name or ID of the user
4	Any text

7.8.5 Commenting out program code

Description

Individual lines or contiguous ranges in robots and submit programs can be commented out. It is also possible to uncomment text again.

- Program text that has been commented out is automatically displayed with light gray text.

  (>>> *7.8.6 "Display of comments in the program" Page 355*)
- An inline form that has been commented out by means of **Comment out** cannot be modified. This is only possible once it has been uncommented again.
- If an existing comment is commented out, it is then doubly commented out.

  If the program section is uncommented again, the original state is restored, i.e. what was already a comment remains a comment.

Behavior with folds:

If a closed fold is commented out, the entire fold including its contents is commented out.

Opened fold:

- If the cursor is located on the FOLD or ENDFOLD line, the entire fold including its contents is commented out.
- If the cursor is located on a line within the fold, only this line is commented out.

On uncommenting, the original state is restored, i.e. if a closed fold already contains a comment and if this closed fold is commented out and then uncommented again, the original comment is still/again present.



It is not possible to recognize from a closed fold whether or not its contents are commented out.

Precondition

- User rights: function group **General KRL program changes**
- T1 mode
- For individual lines: program is selected or open.
  For ranges: program has been opened.

Procedure

To comment out a line:

1. Place the cursor in the line.
2. Menu sequence **Edit** > **Comment out**

To comment out a range:

1. Position the cursor in the line in which the range is to start.
2. Menu sequence **Edit** > **Mark region**
3. Position the cursor in the line in which the range is to end.
   The range, including the start and end lines, is now selected.
4. Menu sequence **Edit** > **Comment out**

To uncomment a line or range:

- Same procedure, but instead of **Comment out**, select the menu item **Undo commenting**.

7.8.6 Display of comments in the program

Semicolon

In KRL, the semicolon is the character that introduces a comment.

There are simplified displays in which the semicolon is not visible, e.g. most folds with detail view deactivated.

- Comments generated via a menu command receive the semicolon automatically (whether it is visible or not).
- For comments inserted manually, the semicolon must be typed.

Text color light gray

Comments in the program are automatically displayed with light gray text.

While there are exceptions, the following applies: light gray text is always a comment.

Examples

```
 6  OUT 11 '' State=True
 7  WAIT Time= 1.0 sec
 8  ;WAIT SEC 1
 9  ;this is a manually inserted comment
10  ; vacuum off
```

Fig. 7-15: Display of comments in the program

Line	Description
7	**Light gray** (semicolon not visible because detail view is deactivated): inline form that was commented out using the **Comment out** functionality (>>> *7.8.5 "Commenting out program code" Page 354*)
8	**Light gray**: KRL command that was commented out using the **Comment out** functionality
9	**Light gray**: comment entered manually
10	**Dark red** (= color for closed folds): comment that was inserted via **Commands** > **Comment** > **Normal** (>>> *7.8.4 "Inserting a comment or stamp in a program" Page 353*)

7.8.7 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

(>>> *10.7.4 "Transforming blocks of coordinates" Page 503*)

7.9 Archiving and restoring data

7.9.1 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*.*
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

Menu item	Archives the directories/files
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 this diagnosis package. 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. (>>> *13.6 "Automatically compressing data for error analysis (diagnosis package "KRCDiag")" Page 720*)

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.

7.9.2 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.

7.9.3 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.

7.9.4 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.

7.9.5 Restoring data

Description

CAUTION

Risk of injury due to archive from a different KSS version
Only archives from the same KSS version may be loaded into KSS. For example:

- Only KSS 8.6 archives in KSS 8.6
- Only KSS 8.7 archives in KSS 8.7

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.

7.10 Project management

7.10.1 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.

7.10.2 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.
     (>>> *7.10.3.1 "Projects tab" Page 363*)
   - **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.
     (>>> *7.10.3.2 "Restoration points tab" Page 365*)
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

Danger to life and limb due to failure to check EMERGENCY STOP and “Operator safety”
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.
If the safety functions are not checked, this may result in death, severe injuries or damage to property.
- 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, the check must be carried out for every robot controller.

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

Danger to life and limb due to unchecked safety configuration
It is possible that the robot could be operated with incorrect data following activation of a project on the robot controller. Death, serious injuries or major damage to property may result.
- Following activation of a WorkVisual project, check the safety configuration on the robot controller.

(>>> *6.6 "Checking the safety configuration of the robot controller" Page 245*)

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.

7.10.3 Project management window

The **Project management** window is opened using the WorkVisual icon on the smartHMI.

7.10.3.1 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.

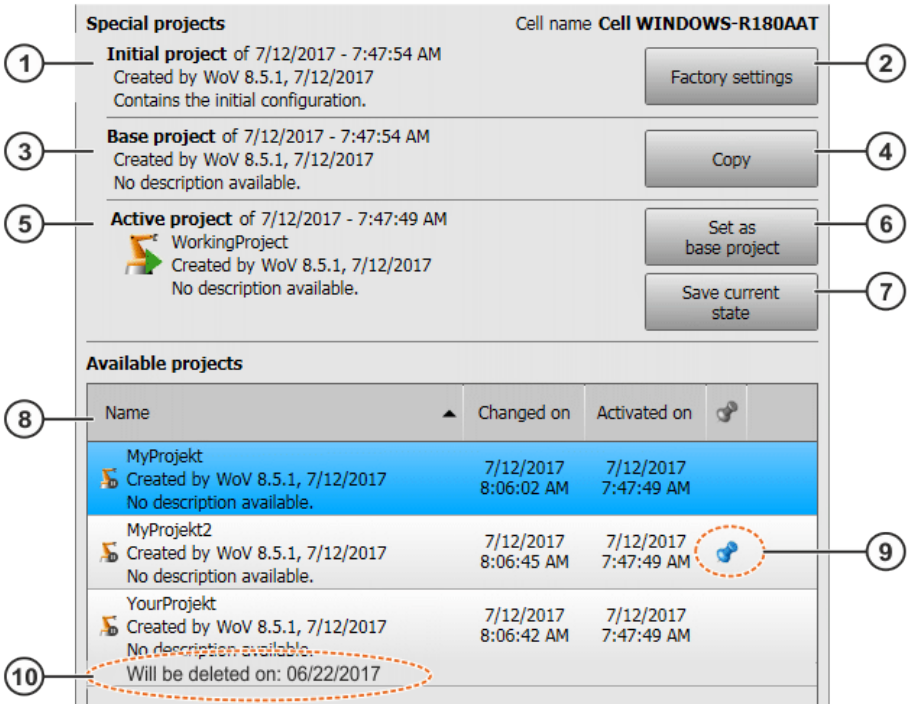


Fig. 7-16: Projects tab

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. (>>> *7.10.1 "Pinning a project on the robot controller" Page 361*) 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.

7.10.3.2 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.

7.11 Backup Manager

7.11.1 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, 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	(>>> *7.11.2 "Backing up projects, option packages and RDC data manually" Page 367*) (>>> *7.11.3 "Manually restoring projects and option packages" Page 368*) (>>> *7.11.4 "Restoring RDC data manually" Page 370*)
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.

7.11.2 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.

7.11.3 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

Danger to life and limb due to failure to check EMERGENCY STOP and “Operator safety”
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.
If the safety functions are not checked, this may result in death, severe injuries or damage to property.

- 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, the check must be carried out for every robot controller.

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

Danger to life and limb due to unchecked safety configuration
It is possible that the robot could be operated with incorrect data following activation of a project on the robot controller. Death, serious injuries or major damage to property may result.

- Following activation of a WorkVisual project, check the safety configuration on the robot controller.

(>>> *6.6 "Checking the safety configuration of the robot controller" Page 245*)

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.

7.11.4 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}.*

7.11.5 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**.

7.11.5.1 “Backup configuration” tab

Backup configuration

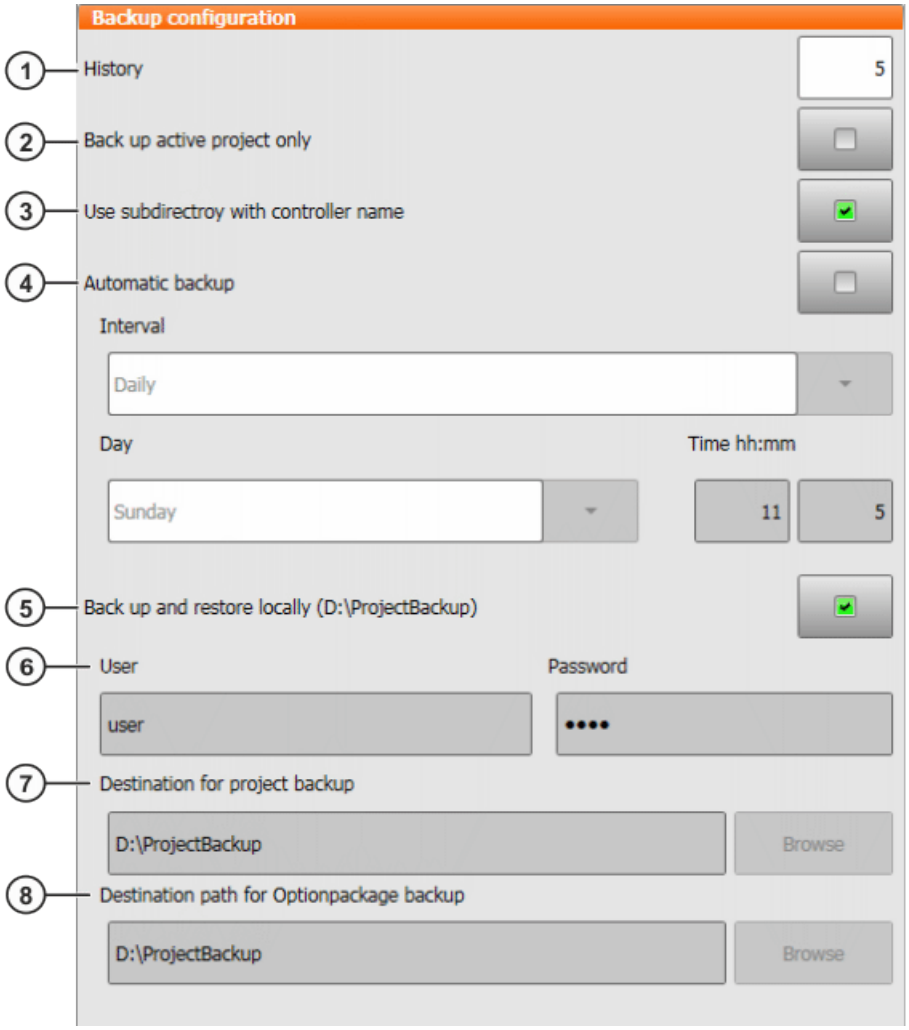


Fig. 7-17: “Backup configuration” tab

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.

7.11.5.2 “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

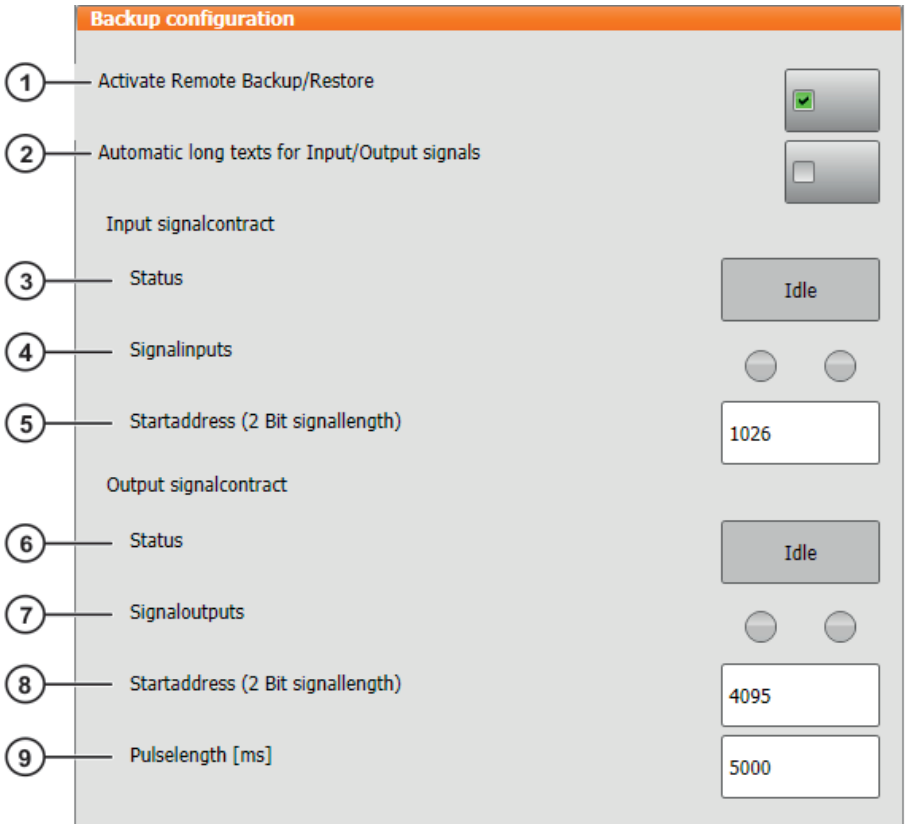


Fig. 7-18: “Signal interface” tab

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 375*)

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 375*)
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 unsuccessful	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.

8.1.1 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 display. 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.

8.2 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.

8.3 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

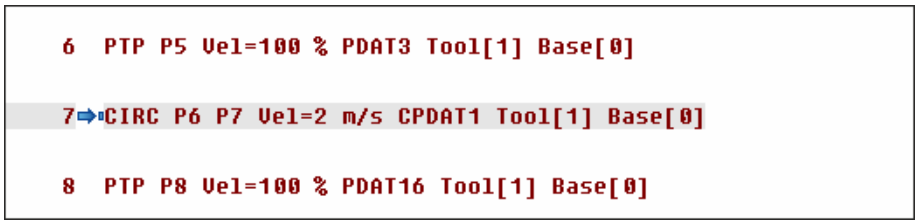


Fig. 8-5: The robot has reached auxiliary point P6 with exact positioning

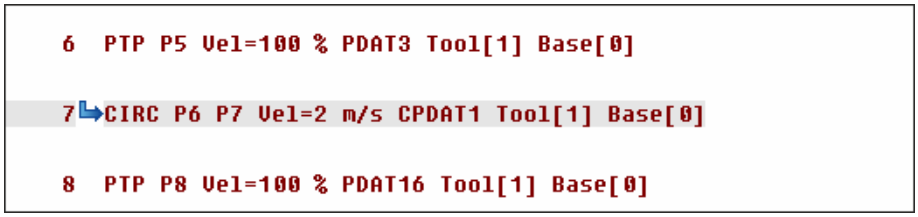


Fig. 8-6: The robot is moving from auxiliary point P6 to P7

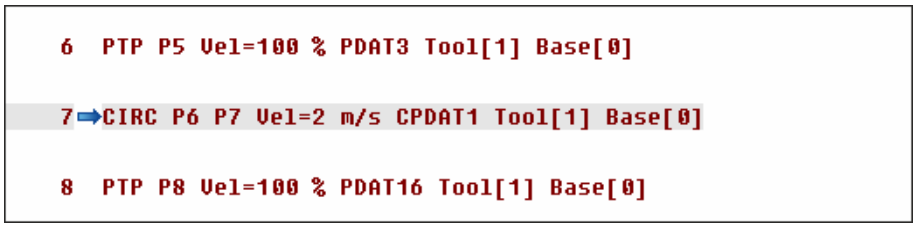


Fig. 8-7: The robot has reached P7 with exact positioning

Examples for backward motion

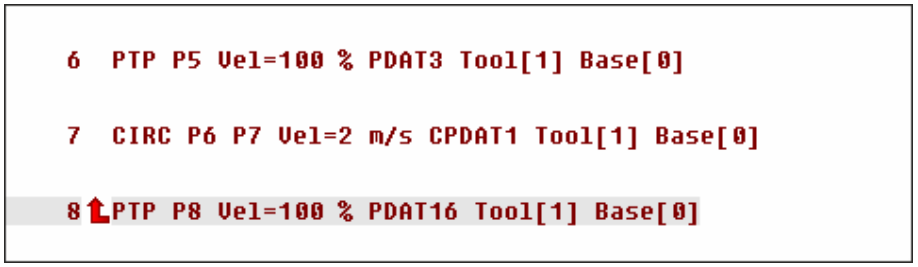


Fig. 8-8: The robot is moving from P8 to P7

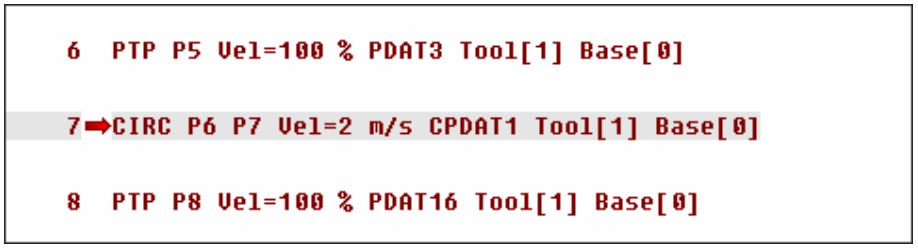


Fig. 8-9: The robot has reached P7 with exact positioning

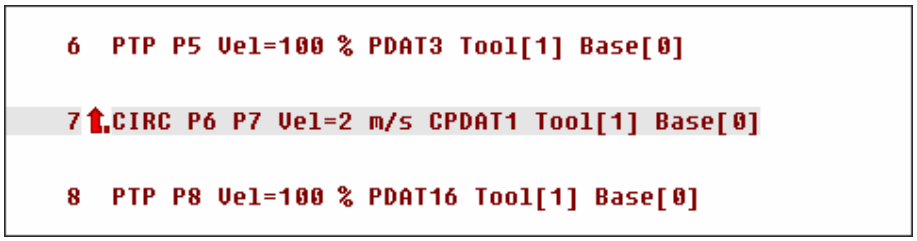


Fig. 8-10: The robot is moving from P7 to auxiliary point P6

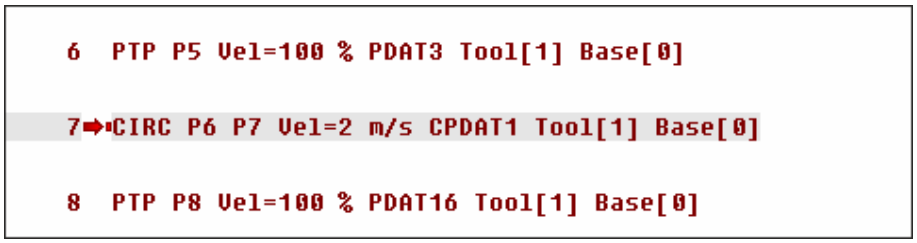


Fig. 8-11: The robot has reached auxiliary point P6 with exact positioning

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.

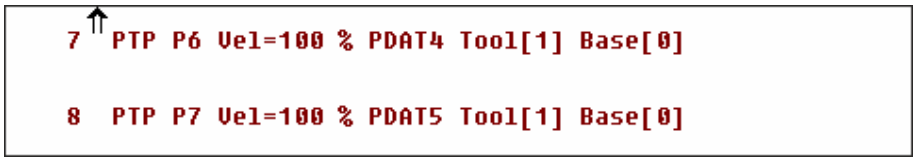


Fig. 8-12: The block pointer is located higher up in the program



Fig. 8-13: The block pointer is located lower down in the program

8.4 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.



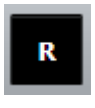
Fig. 8-14: Overrides status indicator

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%.

8.5 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.

8.6 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

Risk of injury due to unexpected path of BCO
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. Failure to take this into account may result in injuries.

– Observe the motion during the BCO run so that it is possible to react 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.

8.7 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

Risk of injury due to unexpected path of BCO
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. Failure to take this into account may result in injuries.

– Observe the motion during the BCO run so that it is possible to react 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.

8.8 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.

8.9 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

Risk of injury due to unexpected path of BCO
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. Failure to take this into account may result in injuries.

– Observe the motion during the BCO run so that it is possible to react 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.

8.10 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**.

8.11 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

Risk of injury due to unexpected path of BCO
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. Failure to take this into account may result in injuries.

– Observe the motion during the BCO run so that it is possible to react 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.

8.12 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:
(>>> *4.17.12 "Backward motion using the jog keys" Page 101*)
An overview of the most important differences can be found here:
(>>> *8.12.4 "Comparison of “Start backwards”/backwards using the jog keys" Page 391*)

8.12.1 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.

8.12.2 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.

(>>> *6.21 "Configuring backward motion" Page 275*)

`DELETE_BACKWARD_BUFFER()` can be used to prevent backward motion for specific motions.

(>>> *11.16.1 "DELETE_BACKWARD_BUFFER: preventing backward start" Page 660*)

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.

8.12.2.1 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.

8.12.2.2 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

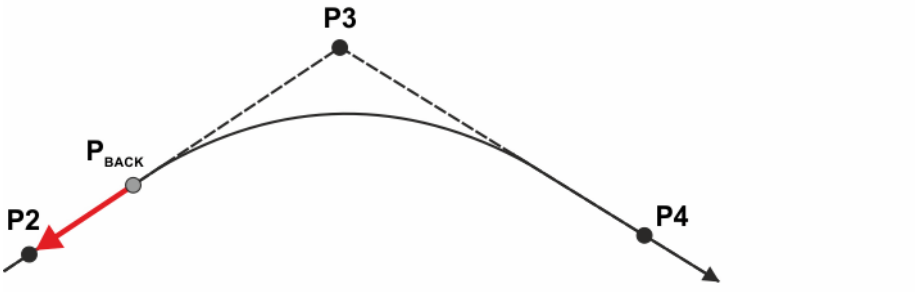


Fig. 8-16: Case 1: Backward start outside an approximate positioning range

If the end point of the previous motion is approximated, it is nonetheless addressed with exact positioning.

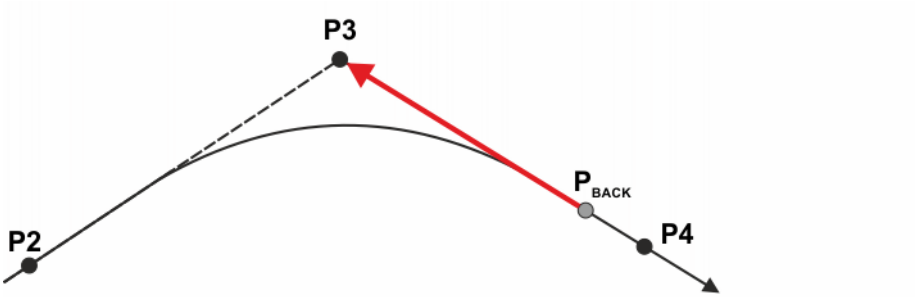


Fig. 8-17: Case 2: Backward start outside an approximate positioning range

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.

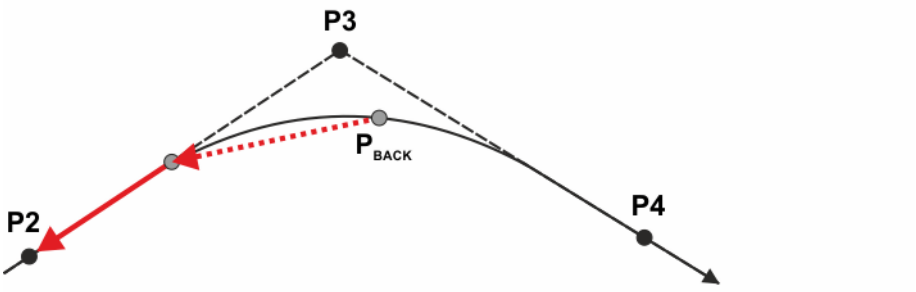


Fig. 8-18: Backward start in the approximate positioning range

8.12.2.3 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

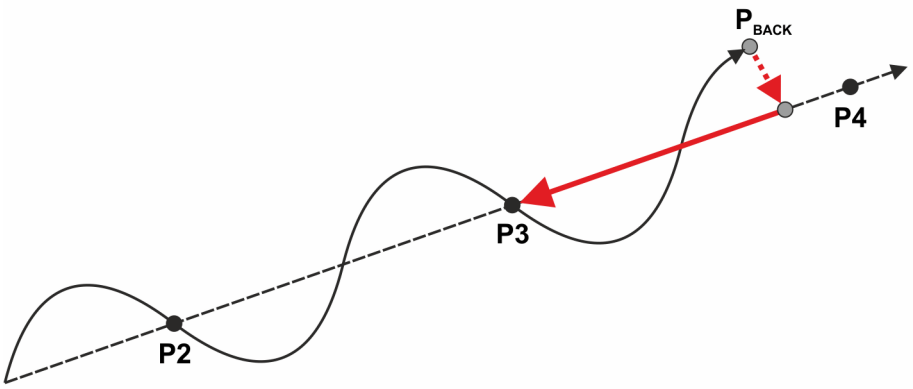


Fig. 8-19: Backward start on weave path

8.12.2.4 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.

8.12.3 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.

8.12.4 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

8.13 Making a subprogram invisible

Description

Modules – in other words, programs – can be made invisible. This means that if they are called as subprograms, they will not be displayed in the editor during program execution. Invisible modules cannot be selected directly.

Precondition

- User rights: Function group **Critical KRL program changes**
- T1, T2 or AUT mode
- The program is not selected.

Procedure

1. Select the SRC file of the desired module in the Navigator.
2. Select **Edit** > **Properties** in the menu.
3. Switch to the **Module info** tab.
4. Deactivate the check box for **Visible** in the **Module attributes:** area.
   This means the module will no longer be visible.



The **Visible** attribute hides the module in the editor during program execution. The similarly named **Hidden** attribute on the **General** tab, on the other hand, hides the file in the file system.

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 exceed a limit value, 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
- $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.

8.14.1.1 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.

8.14.1.2 Determining peak values for collision detection

Description

While the robot is moving, the robot controller continuously measures certain forces and torques, and determines the axis-specific peak values. The peak values are determined separately for jog mode and program mode.

The user has the option of applying the peak values as limit values for collision detection.



The buttons used for applying or resetting the peak values are described in the sections on configuration.

- All peak values can be applied, or a selection, e.g. for specific axes.
- If peak values are applied, the robot controller automatically adds a (configurable) offset.
- The user has the option of resetting the peak values in order to be able to determine them again for specific programs or motions.

Precondition

Values are determined in both jog mode and program mode. Precondition for determining peak values:

- Collision detection is active.
- Alternatively in jog mode: The **Collision detection - Jogging configuration** window is open.

  Main menu **Configuration** > **Collision detection** > **Jogging configuration**

8.14.2 Configuring collision detection for jog mode

Description

Collision detection for jog mode is inactive if the limit values for all axes are “0”. This is the default setting.

In order to use the collision detection function, limit values other than “0” must be set. The values can be determined or entered directly.



Furthermore, in addition to the procedure described here, the sensitivity of the collision detection function can be changed in the **Jog options** window by means of the **Default value offset** setting.

Precondition

- User rights: Function group **Calibration**

Procedure: determining values

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 **Data set values** column.
5. Save.

Procedure: entering values directly

1. In the main menu, select **Configuration** > **Collision detection** > **Jogging configuration**.
   The **Collision detection - Jogging configuration** window opens.
2. Enter the desired limit values manually in the **Data set values** column.
3. Save.

8.14.2.1 “Collision detection - Jogging configuration” window

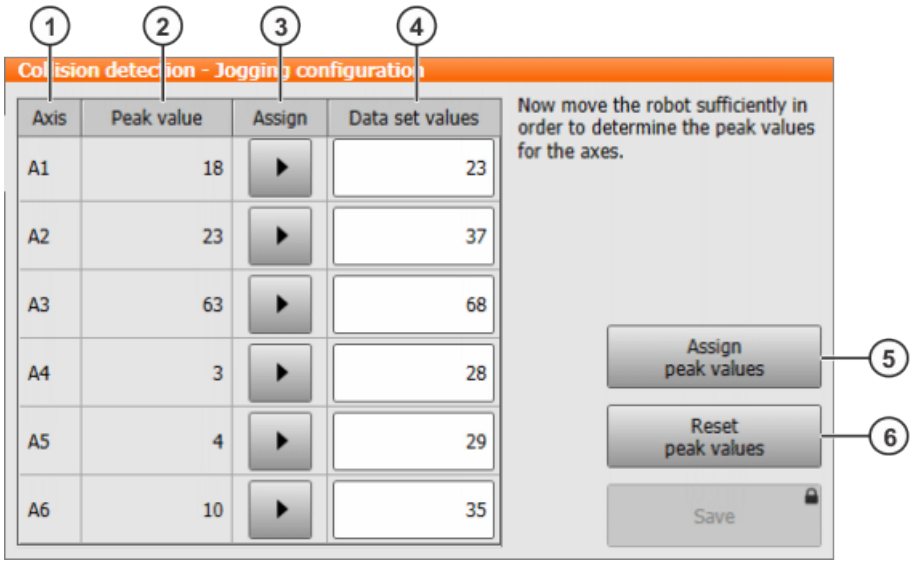


Fig. 8-20: Collision detection - Jogging configuration

Item	Description
1	Axis number
2	The highest value that has occurred since the last reset of the peak values (>>> *8.14.1.2 "Determining peak values for collision detection" Page 394*) • 0 … 500
3	Transfers the value from the **Peak value** box to the **Data set values** box and adds the associated offset. (>>> *8.14.4.3 "Defining offsets / Offset settings window" Page 399*)
4	Limit 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 **Data set values** column and adds the associated offset.
6	Sets the peak values for all axes to zero for jog mode.

8.14.3 Configuring collision detection for program mode – overview

There are 30 data sets available for program mode. In the factory settings, the data sets contain a default value for collision detection.

The following options are available for filling the data sets with other suitable values:

- Via the corresponding configuration window:
  Apply values determined by the robot controller or enter values directly
  (>>> *8.14.4 "Filling data sets via the configuration window" Page 396*)
- Via a program run in teach mode
  (>>> *8.14.5 "Filling data sets via a program in teach mode" Page 400*)

8.14.4 Filling data sets via the configuration window

Description

This procedure can be used to fill the data sets for the limit values individually.

- Either enter values manually.
- Or accept values determined by the robot controller.

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 overview** window opens.
   (>>> *8.14.4.1 ""Collision detection - Data set overview" window" Page 397*)
2. Mark the desired data set and select **Edit**.
   The **Collision detection - Data set configuration** window opens.
   (>>> *8.14.4.2 ""Collision detection - Data set configuration" window" Page 398*)
3. Enter a name for the data set or modify the existing name.
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.

8.14.4.1 "Collision detection - Data set overview" window

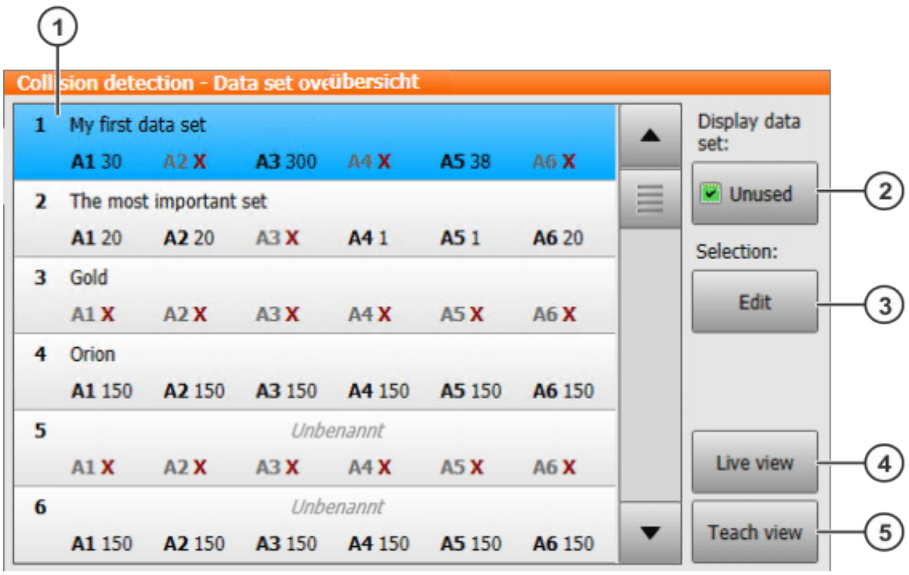


Fig. 8-21: "Collision detection - Data set overview" window

Item	Description
1	The data sets are displayed in a list. A red "X" means: the value for this axis is zero. Collision detection is thus not active for this axis.
2	• Check box active: All data sets are displayed in the list. • Check box not active: The following data sets are not displayed: Data sets for which no name has been assigned yet AND whose axes all have the default value 150.
3	Opens the selected data set in the **Collision detection - Data set configuration** window. Here, the data set can be edited. (>>> *8.14.4.2 ""Collision detection - Data set configuration" window" Page 398*)

Item	Description
5	Switches to the **Collision detection - Display** window. (>>> *8.14.8 "Displaying current values / Collision detection - Display window" Page 405*)
6	Switches to the **Collision detection - Data set teach view** window. (>>> *8.14.5.1 ""Collision detection - Data set teach view" window" Page 401*)

8.14.4.2 "Collision detection - Data set configuration" window

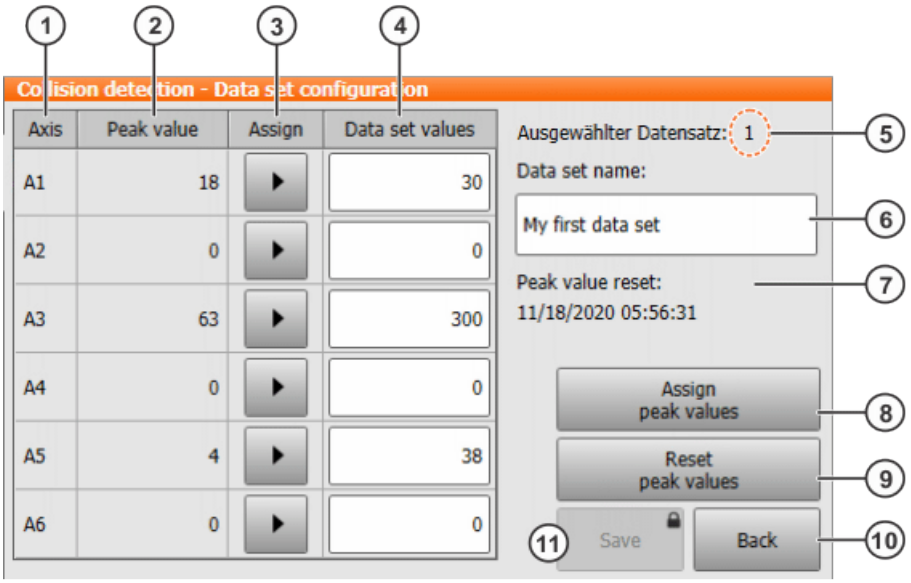


Fig. 8-22: "Collision detection - Data set configuration" window

Item	Description
1	Axis number
2	The highest value that has occurred since the last reset of the peak values. The value is data set-specific. (>>> *8.14.1.2 "Determining peak values for collision detection" Page 394*) • 0 … 500
3	Transfers the value from the **Peak value** box to the **Data set values** box and adds the associated offset. (>>> *8.14.4.3 "Defining offsets / Offset settings window" Page 399*)
4	Limit 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: 150.
5	Number of this data set

Item	Description
6	For this data set, a name can be entered or the existing name can be changed (max. 24 characters).
7	The time at which the peak values were last reset is displayed.
8	Transfers the values from the **Peak value** column to the **Data set values** column and adds the associated offset.
9	Sets the peak values for this data set to zero.
10	Closes the **Collision detection - Data set configuration** window and displays the **Collision detection - Data set overview** window again.
11	Saves the changes without closing the **Collision detection - Data set configuration** window.

8.14.4.3 Defining offsets / Offset settings window

Precondition

- User rights: Function group **Calibration**

Procedure

1. In the main menu, select **Configuration** > **Collision detection** > **Data set configuration**.
   The **Collision detection - Data set overview** window opens.
   (>>> *8.14.4.1 ""Collision detection - Data set overview" window" Page 397*)
2. Select the **Teach view** button.
   The **Collision detection - Data set teach view** window opens.
   (>>> *8.14.5.1 ""Collision detection - Data set teach view" window" Page 401*)
3. Select the **Offset settings** button.
   The **Collision detection - Offset settings** window opens.
4. Enter the desired offsets.
5. Save.

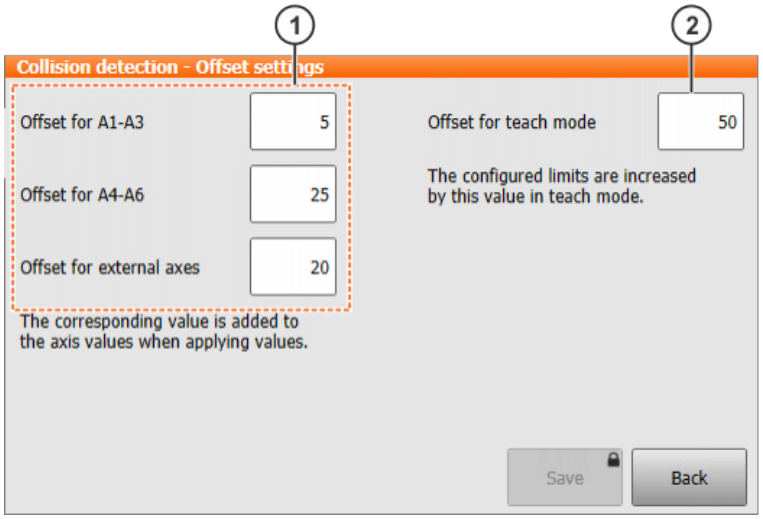


Fig. 8-23: "Collision detection - Offset settings" window

Item	Description
1	If peak values are applied, the controller automatically adds the offset defined here to the peak value. The offsets are valid in jog mode and in program mode.
2	This offset is only valid in teach mode.

8.14.5 Filling data sets via a program in teach mode

Description

The data sets for the limit values can be filled with suitable values via teach mode. This is done by executing the program repeatedly with collision detection activated. Following the “teaching runs”, the peak values can be applied.

- When teach mode is activated, the robot controller sets all peak values to zero (after a request for confirmation). This ensures that up-to-date peak values are available after the teaching run.
- The “offset for teach mode” is only valid in teach mode. This offset ensures that collision detection is not triggered during the teaching run, even if the data sets still contain unsuitable values (default values or old values).

Precondition

- User rights: Function group **Calibration**
- The program has been created.
  A data set has been assigned to the motions that are to be monitored: Either via the inline form **UsaDataSet** or directly at the individual motion.

Procedure

Activate teach mode:

1. In the main menu, select **Configuration** > **Collision detection** > **Data set configuration**.
   The **Collision detection - Data set overview** window opens.
   (>>> *8.14.4.1 "“Collision detection - Data set overview” window" Page 397*)
2. Select the **Teach view** button.
   The **Collision detection - Data set teach view** window opens.
   (>>> *8.14.5.1 "“Collision detection - Data set teach view” window" Page 401*)
3. Activate the **Teach mode** check box.
4. The following confirmation prompt is displayed: *All previously recorded data will be reset. Do you want to activate teach mode now?*
   To activate teach mode, answer the request for confirmation with **Yes**.
5. Save.

Teach and test:

1. Execute the program 10 to 20 times with collision detection activated.
2. In the **Collision detection - Data set teach view** window, check the peak values and apply them partially or in full.
   (>>> *8.14.5.1 "“Collision detection - Data set teach view” window" Page 401*)

3. Deactivate teach mode.
4. Test the program.
5. Perform further teaching runs if required and/or adapt the data sets individually until the test results are satisfactory.

8.14.5.1 “Collision detection - Data set teach view” window

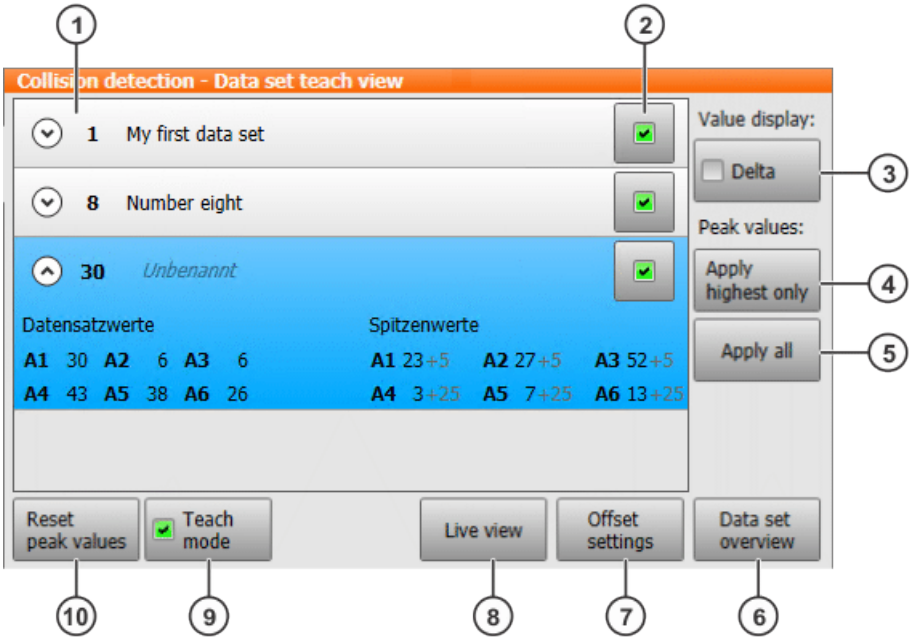


Fig. 8-24: “Collision detection - Data set teach view” window

Item	Description
1	The data sets of the current program are displayed in a list. A data set can be opened by touching the icon. The data set values and the determined peak values are displayed. (>>> *8.14.1.2 "Determining peak values for collision detection" Page 394*)
2	If peak values are only to be applied for some data sets instead of for all of them: Deactivate the check box here for all data sets for which peak values are NOT to be applied.

Item	Description
3	The effect of **Delta** is only visible in data sets that have been expanded. • Check box not active: The **Peak values** display indicates the measured peak values. Additionally, the offset for each peak value is also displayed. • Check box active: **Peak values** shows the difference between the data set value and the corresponding measured peak value. **Example:** Data set value: **A1** 30. Measured peak value: 23. Let the offset be 5. • **Peak values** display without **Delta** check mark: **A1** 23+5 • **Peak values** display with **Delta** check mark: **A1** -7
4	Only applies as data set values those peak values which, including the offset, are higher than the previous data set values.
5	Applies the peak values plus offset as data set values.
6	Switches to the **Collision detection - Data set overview** window. (>>> *8.14.4.1 ""Collision detection - Data set overview" window" Page 397*)
7	Switches to the **Collision detection - Offset settings** window. (>>> *8.14.4.3 "Defining offsets / Offset settings window" Page 399*)
8	Switches to the **Collision detection - Display** window. (>>> *8.14.8 "Displaying current values / Collision detection - Display window" Page 405*)
9	• Check box active: Teach mode is active. This means the teach mode offset applies. **Note**: When the check box is activated, the following request for confirmation appears: *All previously recorded data will be reset. Do you want to activate teach mode now?* • Check box not active: Teach mode is not active.
10	Sets all peak values for program mode and jog mode to zero.

8.14.6 Activating general collision detection

The advantage of general collision detection is that it needs to be activated only once and not for each motion block.
However, if it is to be ensured that collision detection is always active with the defined settings even after block selection, it must be activated for each motion block.
(>>> *8.14.7 "Activating collision detection for a motion" Page 403*)

Precondition

- User rights: Function group **General KRL program changes**
- Program is selected or open.
- T1 mode
- The data set that is to be used has been filled with values.

Procedure: Activation

1. Select **Commands** > **Motion parameters** > **Collision detection**.
2. In the inline form, select the entry **UseDataSet**.
3. In the **DataSet** box, select the data set to be used for collision detection.

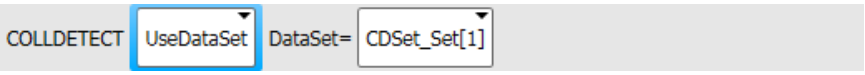


Fig. 8-25

4. Confirm with **Cmd OK**.

Procedure: Deactivation

1. Select **Commands** > **Motion parameters** > **Collision detection**. Select **Off** in the inline form.



Fig. 8-26

2. Confirm with **Cmd OK**.

8.14.7 Activating collision detection for a motion

Description

To activate collision detection for a motion, the user selects a data set in the motion inline form.

Collision detection activated in general and simultaneously for the motion:

General collision detection and collision detection for a motion can be activated simultaneously. The setting for the motion overwrites the general setting.

Spline segments:

Within a spline block, the segments inherit the setting of the block as standard. The user has the option, however, of making individual settings for each segment.

Precondition

- Program is selected or open.
- T1 mode
- The data set that is to be used has been filled with values.

Procedure

1. If the **ColDetect** box 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. 8-27: Inline form with ColDetect, individual SLIN motion 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. 8-28: Inline statement with ColDetect, individual SLIN motion example

Empty box

There is an empty box at the top of the list of data sets. Selecting the empty box means:

- The motion has no separate settings for collision detection. If general collision detection is activated, its settings are inherited.

In spline segments there is no empty box in the list.

If it is desired for a spline segment not to have separate settings, then hide the **ColDetect** box via **Switch parameter** > **Collision detection**. However, “hidden” is already the default state for segments.

OFF

The data set list contains the entry “OFF” for spline segments only. OFF deactivates the collision detection for this segment.

Display of spline settings in the editor

```
SPLINE S1 VEL=2 m/s CPDAT2 Tool[1] Base[1] ColDetect[1]
SLIN P2
SLIN P3 ColDetect[2]
SLIN P4 ColDetect=OFF
ENDPLINE
```

The setting `ColDetect[1]` applies to the block.

SLIN P2 has no separate setting. This means it inherits the setting of the block, namely `ColDetect[1]`.

`ColDetect[2]` applies to SLIN P3.

Collision detection is deactivated for SLIN P4.

8.14.8 Displaying current values / Collision detection - Display window

Procedure

- In the main menu, select **Configuration** > **Collision detection** > **View**.

  The **Collision detection - Display** window opens.

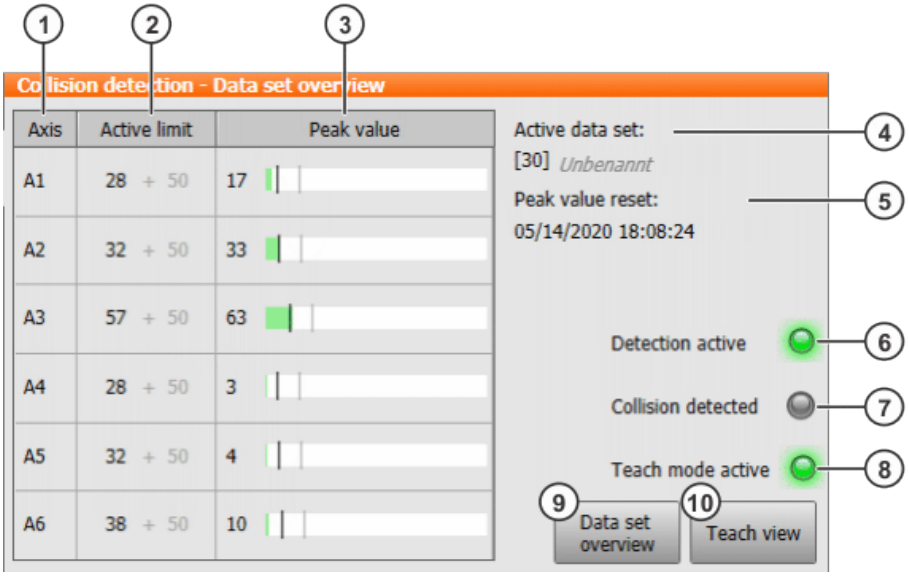


Fig. 8-29: Collision detection - Display (program mode and teach mode)

Item	Description
1	Axis number
2	The current limit for each axis. Depending on the mode in which the robot is being moved, this is either the limit for program mode or the limit for jog mode. The lower the value, the more sensitively the detection responds. “0” means that detection is deactivated for this axis. In teach mode, the teach mode offset is additionally displayed in gray.

Item	Description
3	• **Digits**: current peak value • **Vertical black line**: configured active limit • **Vertical gray line**: effective limit The effective limit can only differ from the configured active limit in teach mode and in jog mode. Teach mode: Limit is determined by the teach mode offset. Jog mode: Limit is determined by the **Default value offset** in the **Jog options** window. • **Colored bar**: Peak value relative to the limit Green: The peak value is below the limit. Red: The peak value has reached or exceeded the limit. A collision has been detected.
4	The data set that is valid for the current motion is displayed. In jog mode, the information **Jogging** is also displayed. BCO runs are always executed with jog mode data. For this reason, **Jogging** is also displayed during and immediately after a BCO run.
5	The time at which the peak values were last reset is displayed.
6	• Gray: Collision detection is not active for the current motion. • Green: Collision detection is active for the current motion.
7	• Red: The controller has detected a collision. • Gray: No collision. If collision detection is not active, the LED is always gray.
8	• Green: Teach mode is active. • Gray: Teach mode is not active. If collision detection is not active, the LED is always gray.
9	Switches to the **Collision detection - Data set overview** window. (>>> *8.14.4.1 ""Collision detection - Data set overview" window" Page 397*)
10	Switches to the **Collision detection - Data set teach view** window. (>>> *8.14.5.1 ""Collision detection - Data set teach view" window" Page 401*)

8.14.9 Inline forms only for compatibility with previous versions

8.14.9.1 Inline form SaveMax

The inline form SaveMax is still available in the KSS for compatibility reasons: In this way, programs from previous KSS versions containing SaveMax can still be used.
For new programs, we urgently recommend using not SaveMax, but the teach mode or the **Collision detection - Data set configuration** window for filling data sets!
Among other things, SaveMax can cause display errors, particularly in the **Collision detection - Display** window.

Call

1. **Commands** > **Motion parameters** > **Collision detection**
2. In the inline form, select the entry **SaveMax**.

Inline form SaveMax



Fig. 8-30: Inline form SaveMax

8.14.9.2 Inline forms SetDefault, SetLimits

The inline forms SetDefault and SetLimits are still available in the KSS for compatibility reasons: This makes it possible for programs from previous KSS versions containing them to still be used.
For new programs, we urgently recommend using the UseDataSet inline form!
Among other things, SetDefault and SetLimits can cause display errors, particularly in the **Collision detection - Display** window.

Call

- **Commands** > **Motion parameters** > **Collision detection**
- In the inline form, select the entry **SetDefault** or **SetLimits**.

Inline form SetDefault

The default values from $custom.dat are used for collision detection.
(>>> *8.14.9.3 "Saving default values for collision detection in $custom.dat" Page 407*)



Fig. 8-31

Inline form SetLimits

Specify the desired sensitivity of collision detection for each axis. Default: 50%

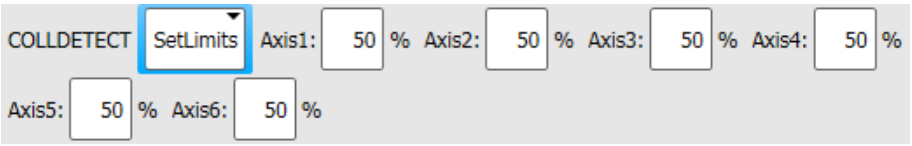


Fig. 8-32

8.14.9.3 Saving default values for collision detection in $custom.dat

Description

If the SetDefault inline form is to be used, the desired values for collision detection must be stored in $custom.dat.

Precondition

- User rights: function group **Critical KRL program changes**
- Suitable values are known that can be used as default values.

Procedure

1. Open the$CUSTOM.dat file under KRC\Steu\MaDa.
2. Set $COLLMON_TOL_PRO_DEF[] to the desired value for the relevant axes.
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.

It is not possible to modify $COLLMON_TOL_PRO_DEF[] via the variable display or by writing to the variable from the program.

8.14.10 Program example: search run via collision detection

Description

The example shows how the collision detection can be used for a search run. The robot is to “find” an object (e.g. a part) and then execute a reaction programmed by the user.

The reactions actually belonging to the collision detection (stop, acknowledgement message, etc.) are not desired in this case and are bypassed.

Basic principle

The peak values arising during the search run are compared to a motion over the same path without collision. If higher peak values occur during the search run, this indicates a collision with the object, i.e. the object has been “found”.

Implementation in the program

The comparison is implemented by means of a cycflag.

If the cycflag detects higher peak values (in other words, a “collision”, i.e. an object has been found), the subprogram with the reaction programmed by the user is called via an interrupt.

Teaching run

Before the actual search, a teaching run is used to determine the peak values for motion without a collision.

- Recommendation: Warm up the robot before the teaching run.
- The data set that is used (no. 29 in the example) must be filled with high values to start with, e.g. with the default value 150. This makes sure that the collision detection is not triggered during the teaching run.

8.14.10.1 Main program

Declarations by the user in DemoProg.dat:

```
DECL INT SearchSensitivity[6]
DECL INT SearchDataSetID
DECL BOOL LEARN
```

DemoProg.src:

```
  DEF DemoProg()
  INI
1 SearchDataSetID=29
2 LEARN=TRUE
3 INTERRUPT DECL 25 WHEN $CYCFLAG[255] == TRUE DO RetreatWhenFound()
4 $CYCFLAG[255]=FALSE
5 BAS(#VEL_PTP,20)
6 SPTP XHOME
7 SPTP_REL{A5 45}
8 Search(LEARN)
  END
```

Line	Description
1	Define which data set of the collision detection (COLLMON_SETTING[]) is to be used for the search run, e.g. data set 29.
2	Define whether the search program Search() is performed for teaching (TRUE) or for searching (FALSE). The value is transferred when the search program is called.
3	If a limit value from data set 29 is exceeded, the subprogram “RetreatWhen-Found” is executed. (>>> *8.14.10.3 "Return motion program" Page 411*) The value comparison is implemented in the search program via $CYCFLAG[255].
4	Set $CYCFLAG[255] to FALSE.
5	In line 6, motion should not be executed at maximum velocity. That is why the velocity is limited here.
6	In many cases, it is sensible to move the robot to a defined position before the search run. In this example, it is moved to XHOME.
7	In the previous line, the robot has been moved to XHOME. Therefore A5 is moved out of the singularity here.
8	Call the search program. (>>> *8.14.10.2 "Search program" Page 409*)

8.14.10.2 Search program

Description

During the search run, it is not desirable for the collision detection to be triggered and for the corresponding reactions (stop, acknowledgement message, etc.) to occur. To prevent this, a different data set is activated for the actual collision detection (no. 30 in the example).

It must be ensured that this other data set is filled with values that are high enough to prevent the collision detection from being triggered as soon as an object is “found”.

Alternatively, the collision detection can be set to zero for the search run and thus deactivated. Use USE_CD_PARAMS(0) for this. Advantage: No further data set has to be reserved. Disadvantage: If errors occur, resulting in an actual collision, this will not be detected.

Search.src:

```
1 DEF Search(LEARN:IN)
2 DECL BOOL LEARN
```

```
 3 DECL INT counter
 4
 5 FOR counter=1 to 6 STEP 1
 6   $COLLMON_MAX[counter]=0
 7 ENDFOR
 8
 9 IF LEARN THEN
10   USE_CD_PARAMS(SearchDataSetID)
11 ELSE
12   $CYCFLAG[255]=((($COLLMON_MAX[1]>SearchSensitivity[1])
     OR($COLLMON_MAX[2]>SearchSensitivity[2])
     OR($COLLMON_MAX[3]>SearchSensitivity[3])
     OR($COLLMON_MAX[4]>SearchSensitivity[4])
     OR($COLLMON_MAX[5]>SearchSensitivity[5])
     OR($COLLMON_MAX[6]>SearchSensitivity[6]))
     AND $COLL_ENABLE AND $IMPROVED_COLLMON)
13   INTERRUPT ON 25
14   USE_CD_PARAMS(30)
15 ENDIF
16
17 SLIN_REL{X 300}
18 USE_CD_PARAMS(0)
19
20 IF LEARN THEN
21   SAVE_CM_MAX_SETTING(SearchDataSetID,10)
22   READ_CM_SETTING(SearchDataSetID,SearchSensitivity[])
23 ELSE
24   INTERRUPT OFF 25
25   WAIT SEC 0
26 ENDIF
27 END
```

Line	Description
1	LEARN is transferred from the main program to the subprogram.
2, 3	Declaration of the variables
5 … 7	Set $COLLMON_MAX[] to zero so that no old and possibly too high peak values are retained.
9 … 15	IF … ELSE : Depending on the value of LEARN, the program parts for teaching or searching are executed.
10	**Only for teaching runs**: Switch on collision detection; use values from data set 29.
12 … 14	**Only for search runs**
12	$CYCFLAG[255] implements the value comparison: For each axis, the peak value ($COLLMON_MAX[]) occurring during the motion is compared to the limit value from data set 29. If the limit value is exceeded, the Cycflag becomes TRUE. The following points are also checked here: • Whether collision detection is generally active ($COLL_ENABLE) • And whether the required level of collision detection is active ($IMPROVED_COLLMON)
13	Activate the interrupt.

Line	Description
14	Switch on collision detection. Use values from another data set (here, as an example, no. 30). Alternative: Switch off collision detection via USE_CD_PARAMS(0).
17	Simple example of a search path **Note:** For teaching runs, do not place anything on the search path that can be found. The values for a collision-free motion are to be taught.
18	Switch off collision detection.
20 …26	IF … ELSE : Depending on the value of LEARN, the program parts for teaching or searching are executed.
21	**Only for teaching runs**: Write the taught values in data set 29, adding an offset of 10. SAVE_CM_MAX_SETTING(*ID*,*offset*): The function writes the peak values that occur during the motion into the data set *ID*; *ID* = IN parameter; *offset* = IN parameter
22	**Only for teaching runs**: Read the values from data set 29 and write it into the SearchSensitivity field. READ_CM_SETTING(*ID*,*values*[]): The function reads COLLMON_SETTING[*ID*]; *ID* = IN parameter; *values* = OUT parameter
24	**Only for search runs**: Deactivate the interrupt.
25	**Only for search runs**: Stop the advance run because of the possible use of RESUME in the return motion program 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.

8.14.10.3 Return motion program

Description

The user can program the desired reaction. In this example, it is desired for the robot to return to the start point of the interrupted motion.

RetreatWhenFound.src:

```
   DEF RetreatWhenFound( )
1  BRAKE
2  WAIT SEC 2
3  PTP $POS_BACK
4  WAIT SEC 2
5  ;RESUME
   END
```

Line	Description
1	Stop the robot. If necessary, BRAKE F or BRAKE FF can also be used.
2	Short wait time, making it possible to check whether the robot has actually stopped.
3	The robot returns to the start position of the interrupted motion.
4	Short wait time, making it possible to check whether the robot has actually stopped.
5	Uncomment this line if necessary. (>>> *11.11.5 "RESUME: cancelling interrupt programs" Page 632*)

8.14.11 $COLLMON_STARTUP_MAX: adapting the inactive collision detection phase after start of motion

Description

When starting to move, the axes must apply torques that cannot be safely predicted due to static friction effects. To prevent the collision detection function from evaluating these torques as a collision, it is automatically deactivated for the motion start. At the end of the starting phase, collision detection is automatically reactivated.

The robot controller considers the starting phase to be completed when a specific velocity, namely the standstill monitoring limit, is exceeded. If the robot moves very slowly, the axes might not reach this limit. The motion is thus still regarded as the starting phase. Collision detection remains inactive.

$COLLMON_STARTUP_MAX limits this inactive collision detection phase at the start of motion.

Precondition

$COLLMON_STARTUP_MAX is only relevant if the following preconditions are met:

- $IMPROVED_COLLMON = TRUE
- Collision detection is activated.

Writability

The variable can be modified in $custom.dat in KRC:\STEU\Mada.

It cannot be modified via the variable display function or program.

Syntax

`$COLLMON_STARTUP_MAX` = *time*

Explanation of the syntax

Element	Description
time	Type: INT Maximum duration (ms) of the inactive collision detection phase at the start of motion. Collision detection is then reactivated, even if the axes are still in the starting phase. • **≥ 50**: Value is used as entered. • **1 … 49**: Value is implicitly increased to 50. • **≤ 0**: No limitation of the inactive phase. This means that collision detection remains inactive as long as the starting phase is in progress, continuously if necessary. Default: 200



At very low axis velocities, collision detection may generally react very sensitively. If collision detection is now reactivated after a certain time due to $COLLMON_STARTUP_MAX (i.e. with any value > 0), it may be triggered undesirably.
Possible remedy:

- Set $COLLMON_STARTUP_MAX = 0.
- Or: Significantly increase the collision detection limit values for the axes concerned.

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 415*)
- **Linear motions (LIN)**
  (>>> *9.3 "Motion type LIN" Page 416*)
- **Circular motion (CIRC)**
  (>>> *9.4 "Motion type CIRC" Page 416*)
- **Spline motions**
  Spline motions have a number of advantages over conventional PTP, LIN and CIRC motions.
  (>>> *9.7 "Spline motion type" Page 421*)

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.

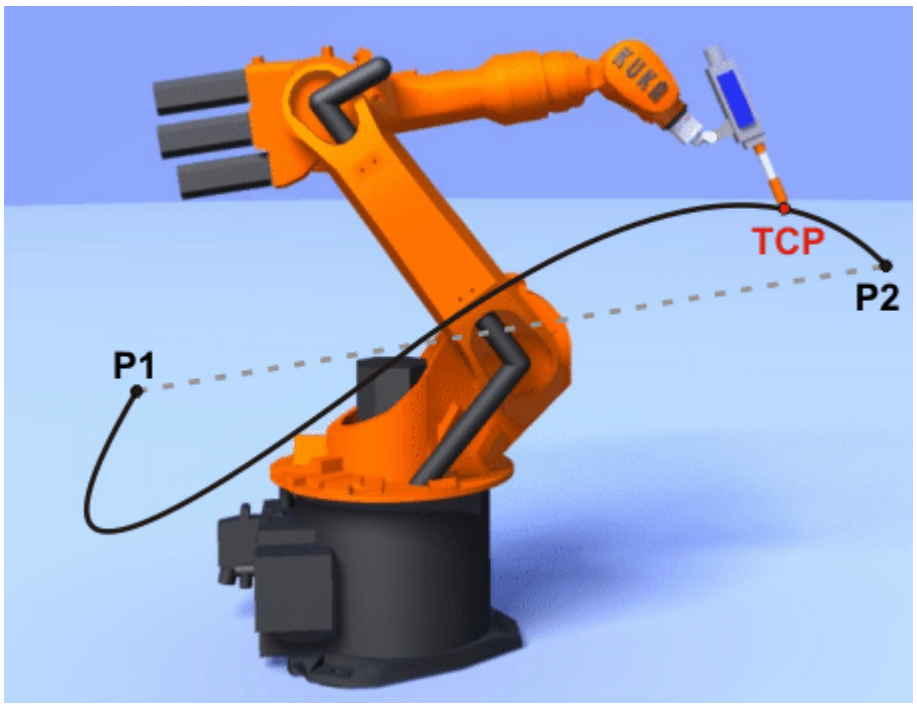


Fig. 9-1: PTP motion

9.3 Motion type LIN

The robot guides the TCP at a defined velocity along a straight path to the end point.

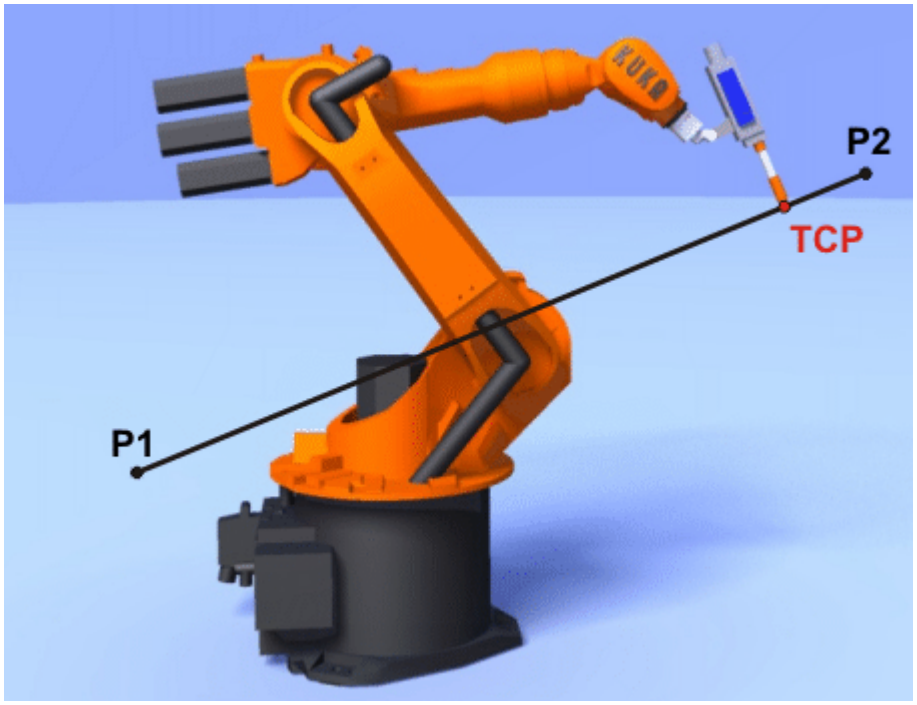


Fig. 9-2: LIN motion

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.

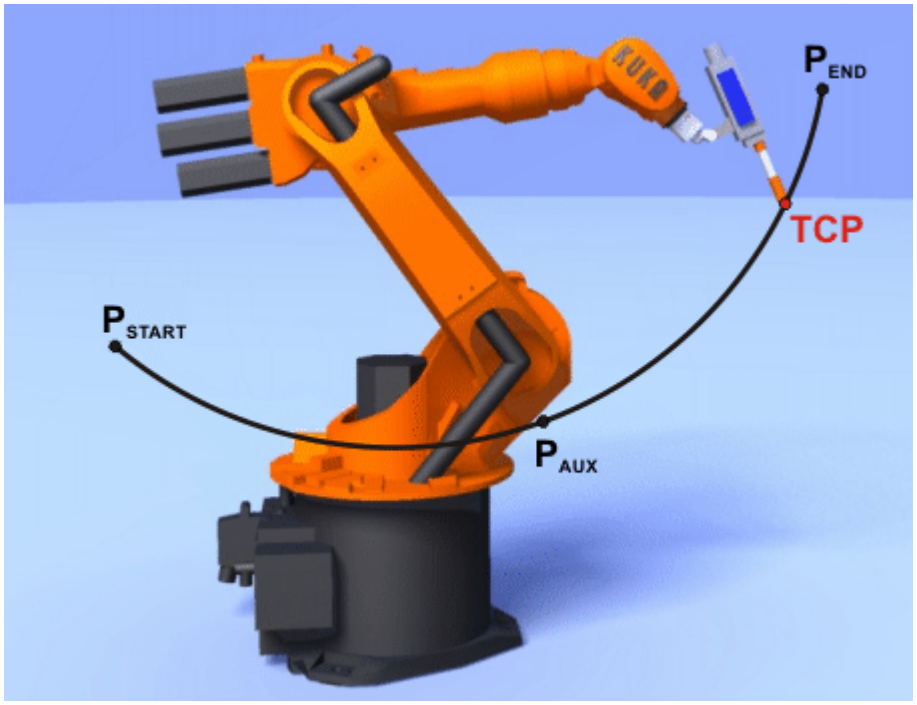


Fig. 9-3: CIRC motion

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.

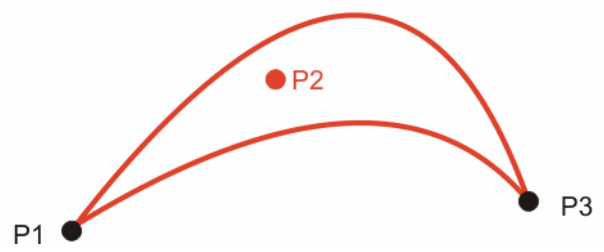


Fig. 9-4: PTP motion, P2 is approximated

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.

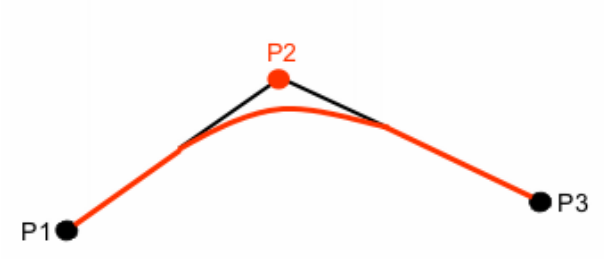


Fig. 9-5: LIN motion, P2 is approximated

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.

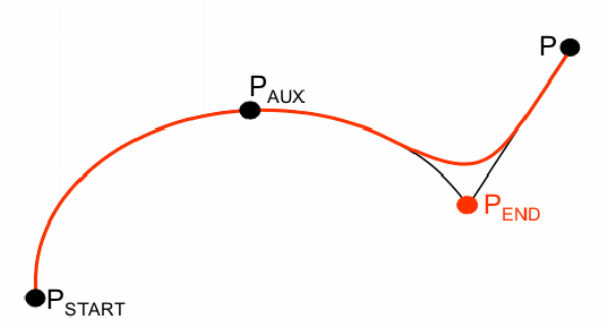


Fig. 9-6: CIRC motion, P_{END} is approximated

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**
  (>>> *10.3.5 "Option window “Motion parameters” (LIN, CIRC, PTP)" Page 456*)
- 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.

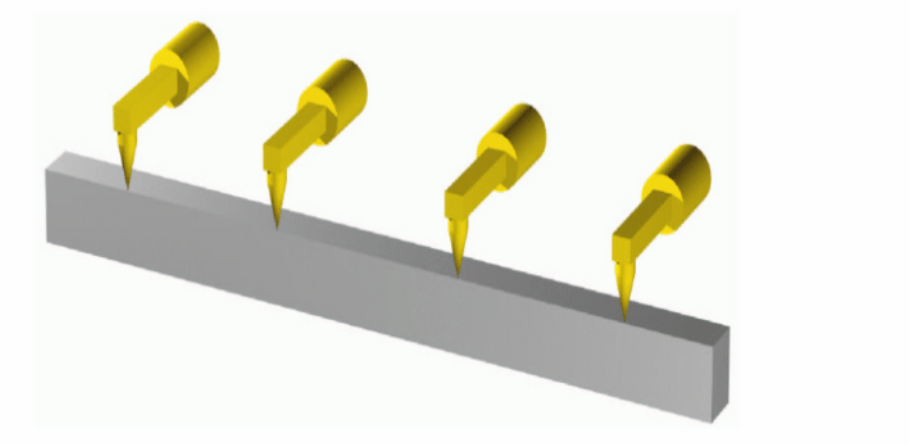


Fig. 9-7: Constant orientation

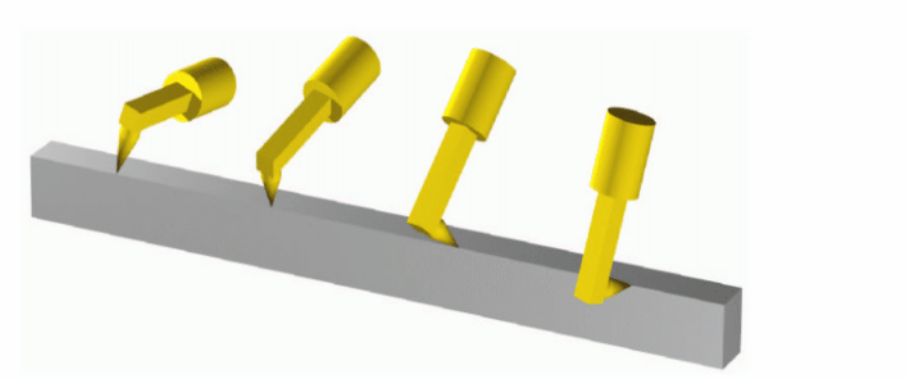


Fig. 9-8: Standard or Wrist PTP

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 419*)

9.6.1 Combinations of $ORI_TYPE and $CIRC_TYPE

$ORI_TYPE = #CONSTANT, $CIRC_TYPE = #PATH:

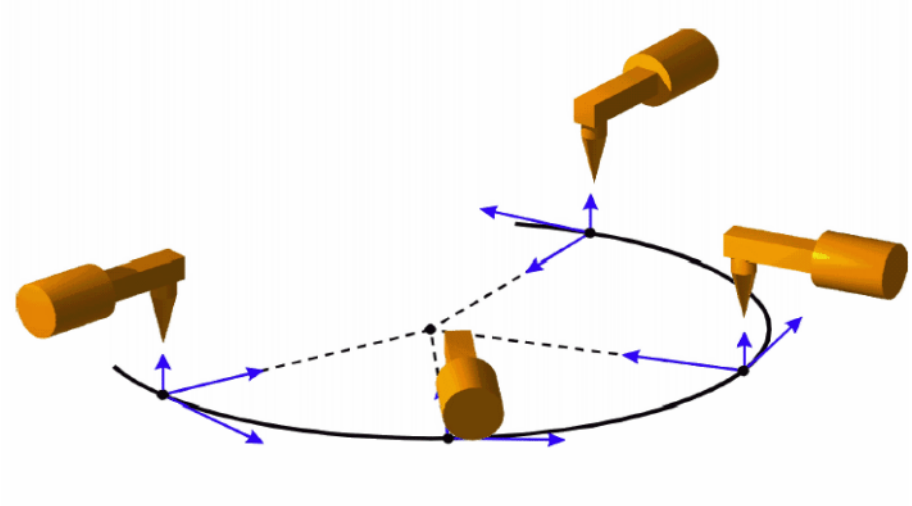


Fig. 9-9: Constant orientation, path-related

$ORI_TYPE = #VAR, $CIRC_TYPE = #PATH:

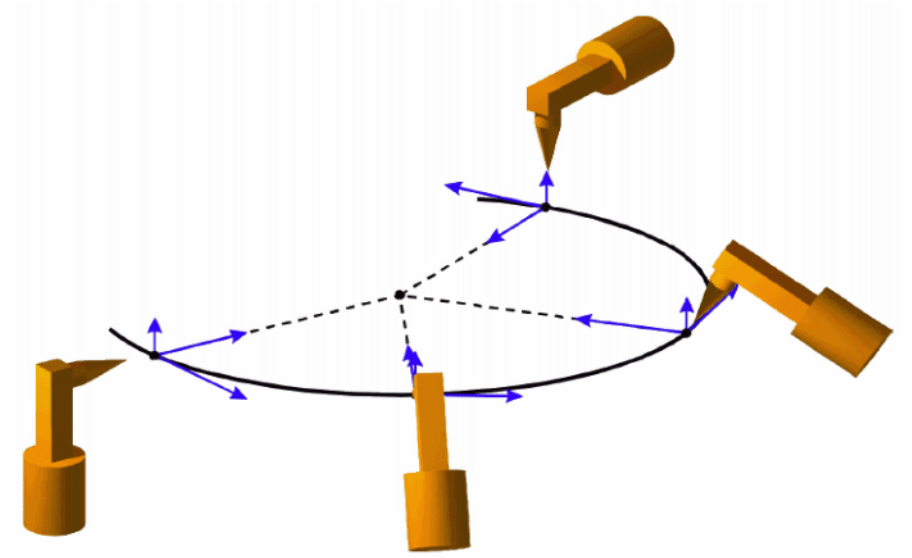


Fig. 9-10: Variable orientation, path-related

$ORI_TYPE = #CONSTANT, $CIRC_TYPE = #BASE:

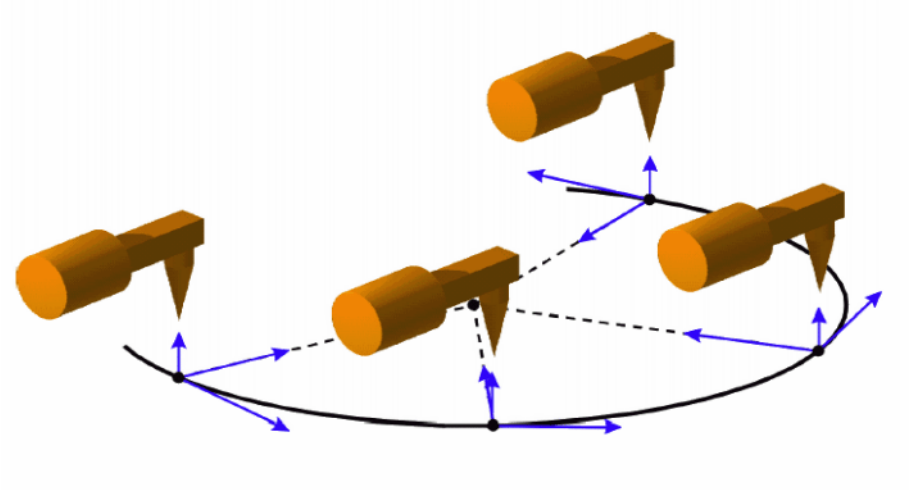


Fig. 9-11: Constant orientation, base-related

$ORI_TYPE = #VAR, $CIRC_TYPE = #BASE:

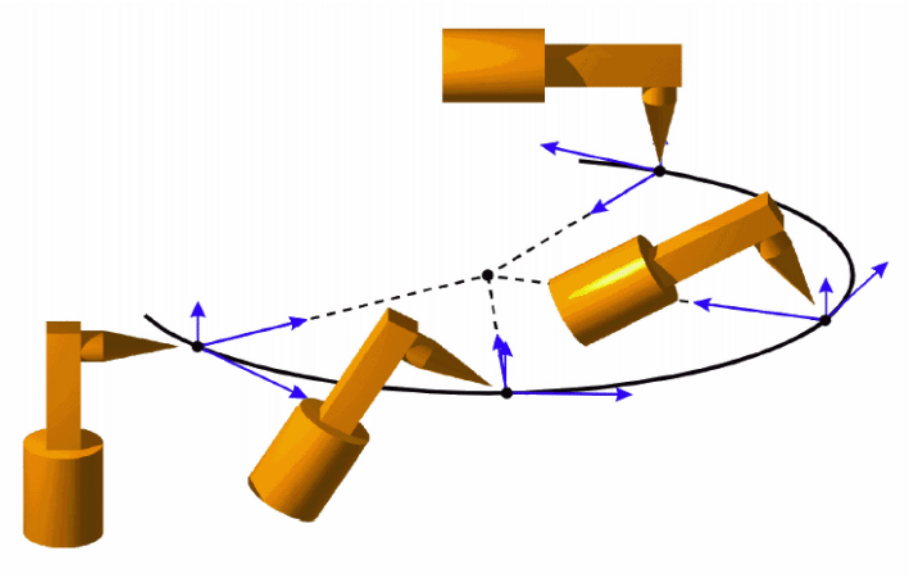


Fig. 9-12: Variable orientation, base-related

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

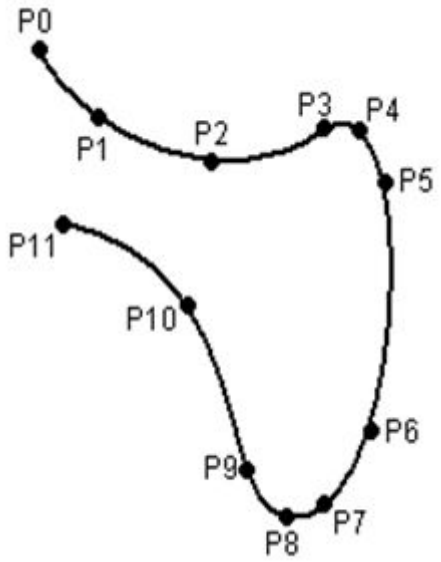


Fig. 9-13: Curved path with spline block

- 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 423*)
  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

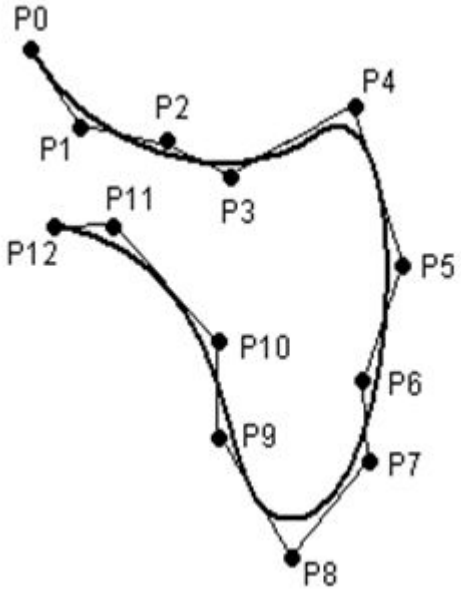


Fig. 9-14: Curved path with approximated LIN motions

- 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.

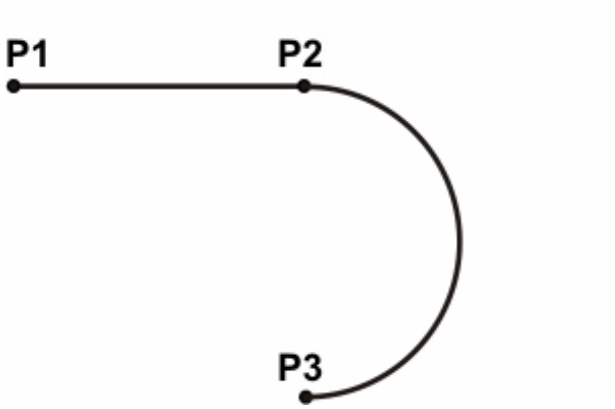


Fig. 9-15: Exact positioning at P2

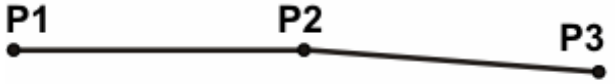


Fig. 9-16: Exact positioning at P2

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.

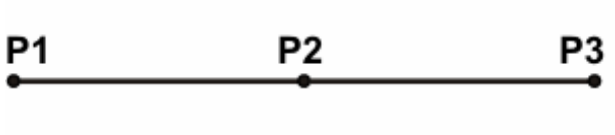


Fig. 9-17: P2 is executed without exact positioning.

- 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.)

i	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

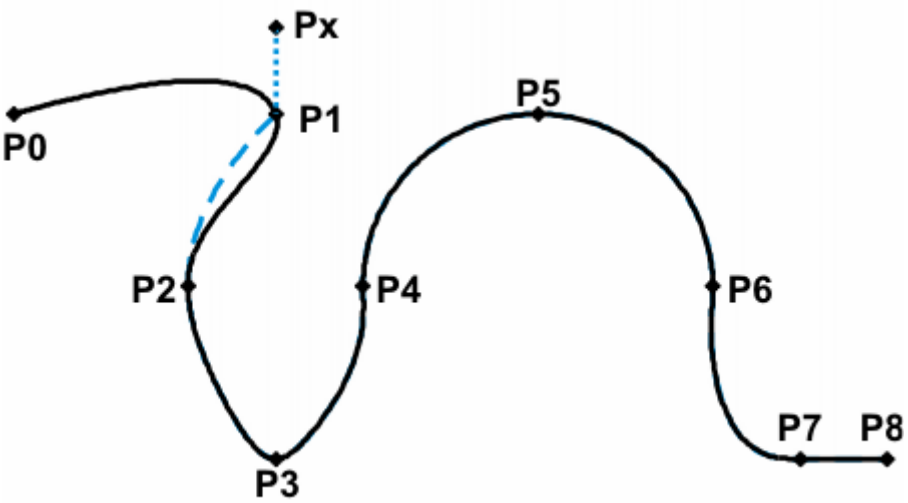


Fig. 9-18: Example: modified path in the case of block selection to P1 with unknown P0

```
 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}*

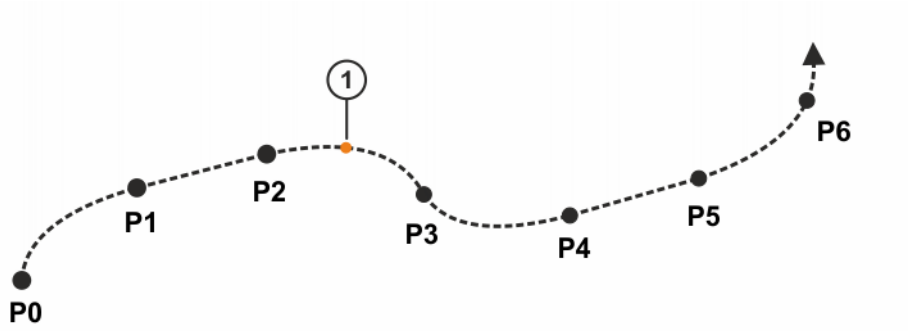


Fig. 9-19: Example

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)*

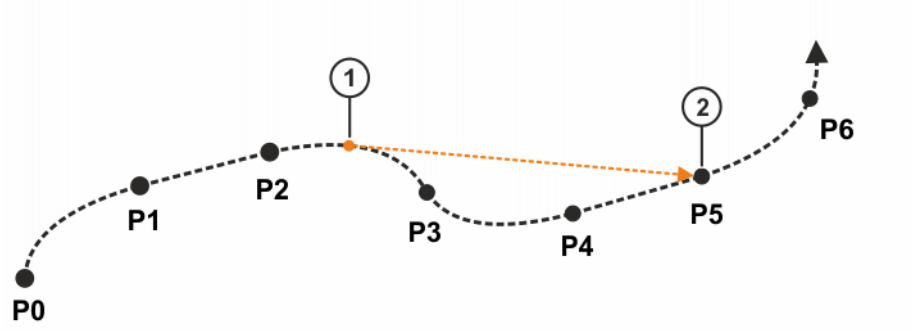


Fig. 9-20: Example 1

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)*

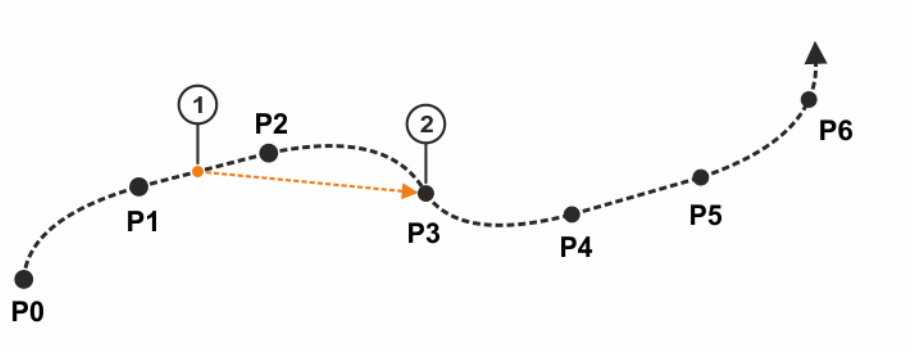


Fig. 9-21: Example 2

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
```

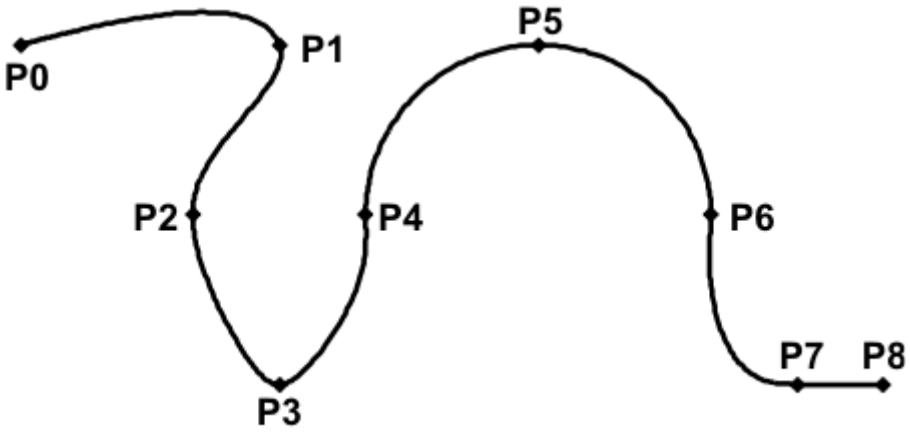


Fig. 9-22: Original path

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.

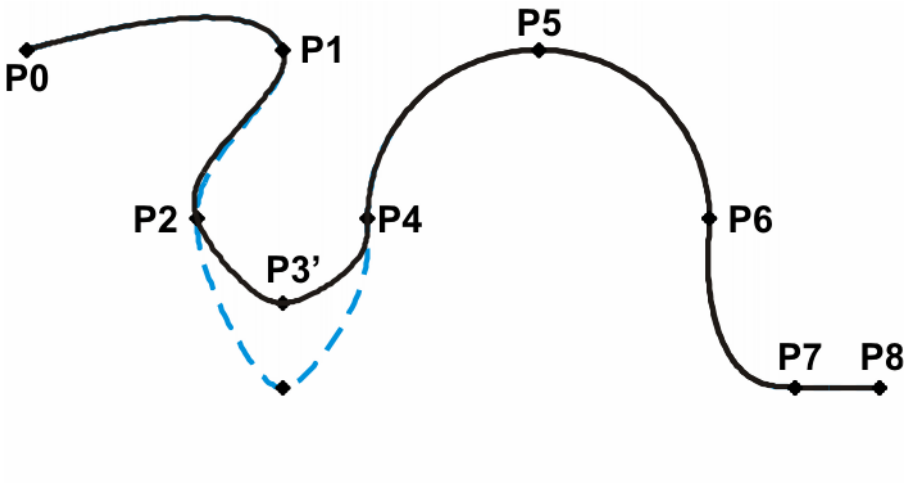


Fig. 9-23: Point has been offset

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
```

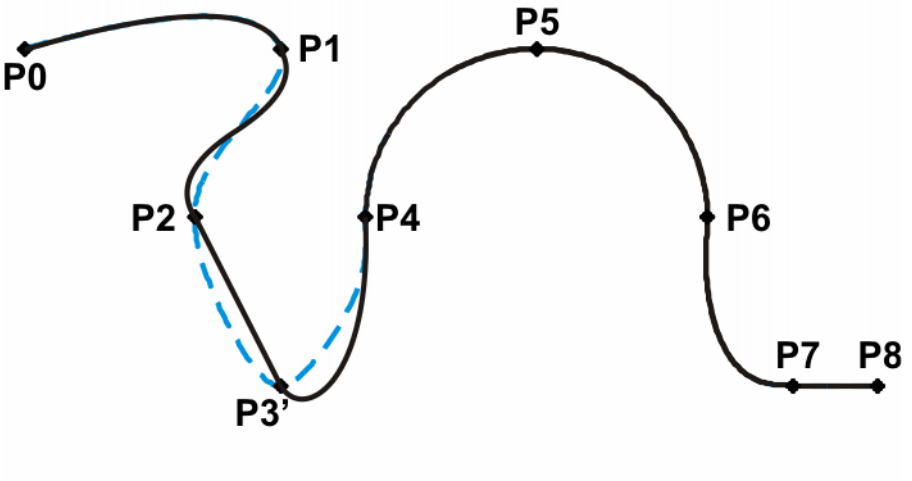


Fig. 9-24: Segment type has been changed

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
```

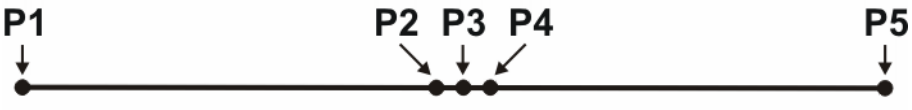


Fig. 9-25: Original path

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
```

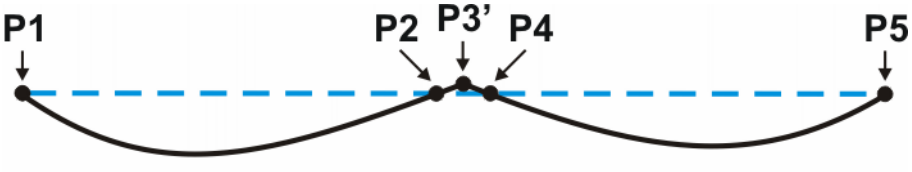


Fig. 9-26: Point has been offset

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

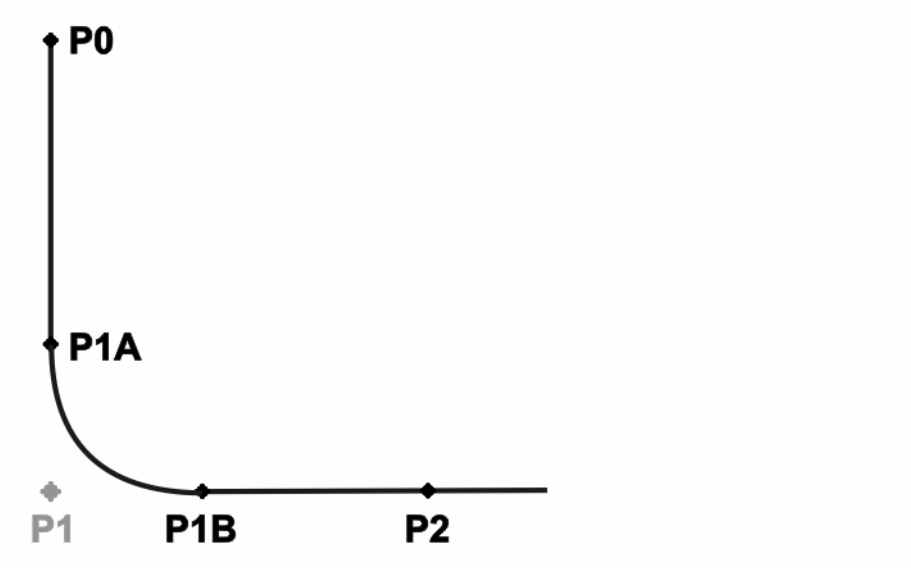


Fig. 9-27: Approximated motion - spline motion

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.

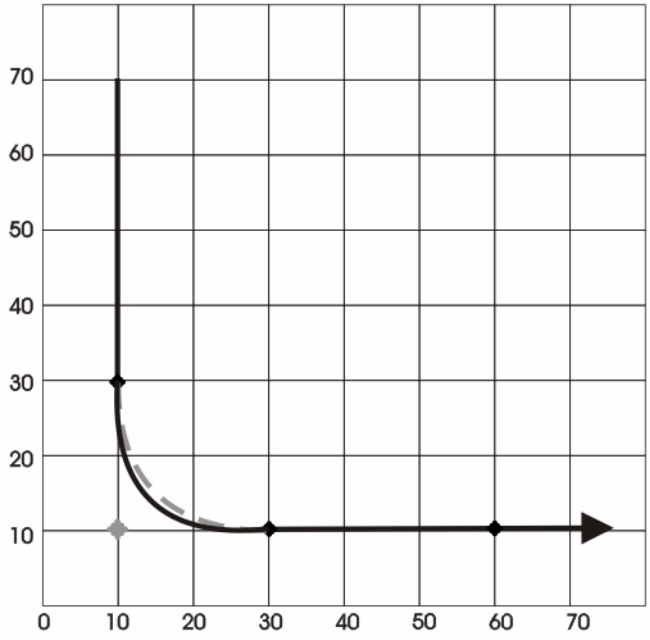


Fig. 9-28: Example: Approximated motion - spline motion 1

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
```

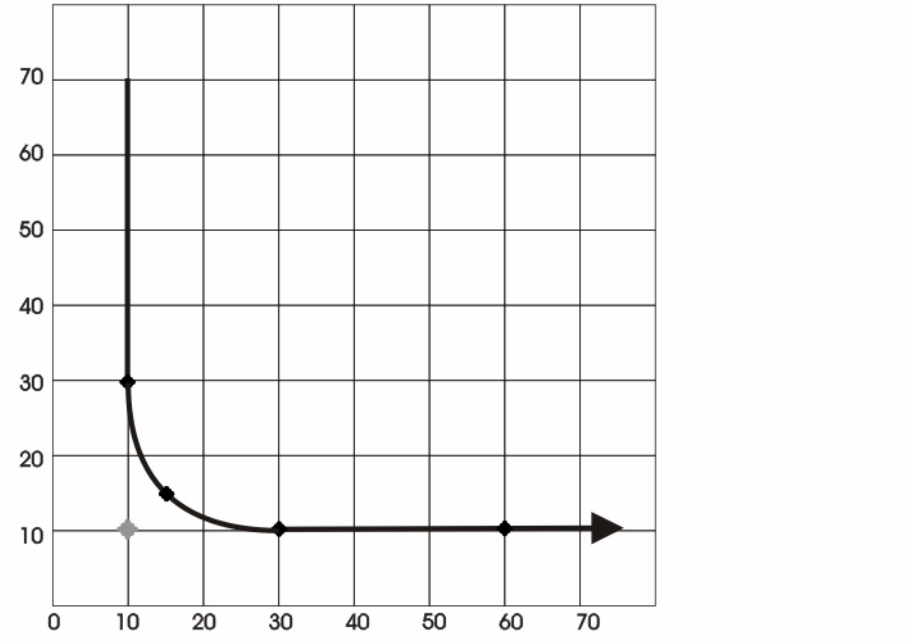


Fig. 9-29: Example: Approximated motion - spline motion 2

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.

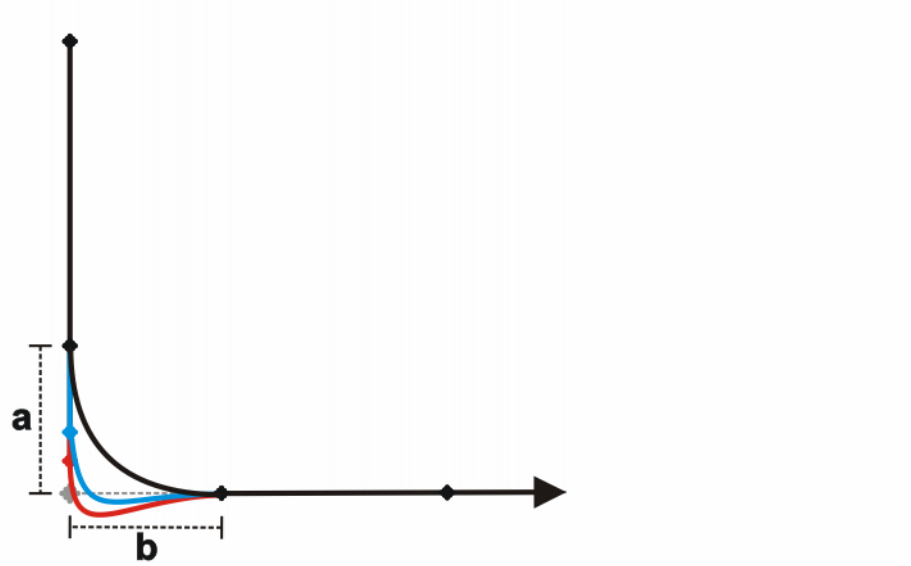


Fig. 9-30: SLIN-SPL-SLIN

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 437*)

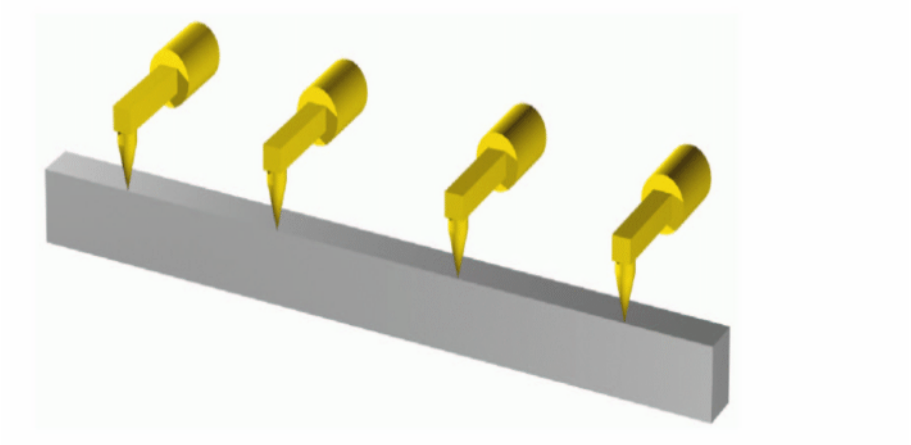


Fig. 9-31: Constant orientation

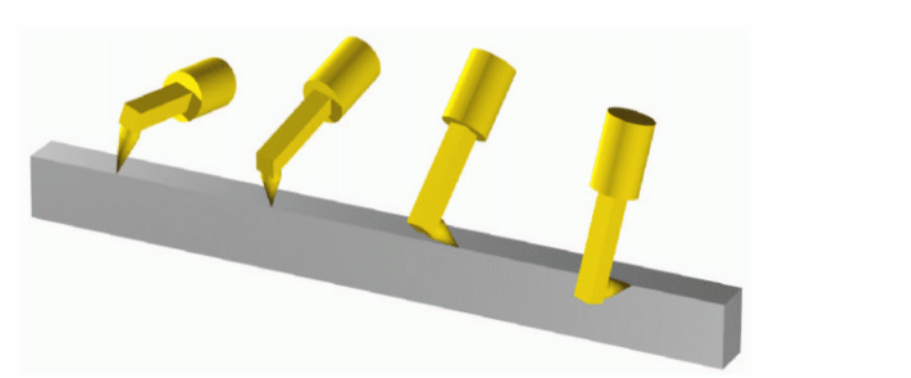


Fig. 9-32: Default orientation control

#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 438*)

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 439*)

$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 419*)

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°.

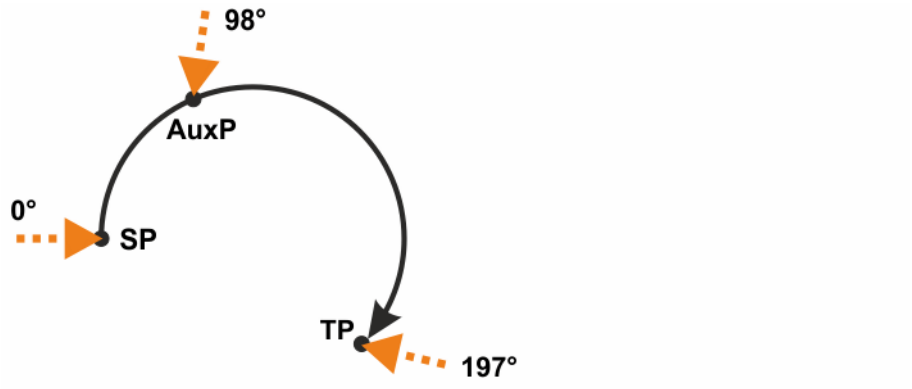


Fig. 9-33: #INTERPOLATE

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.

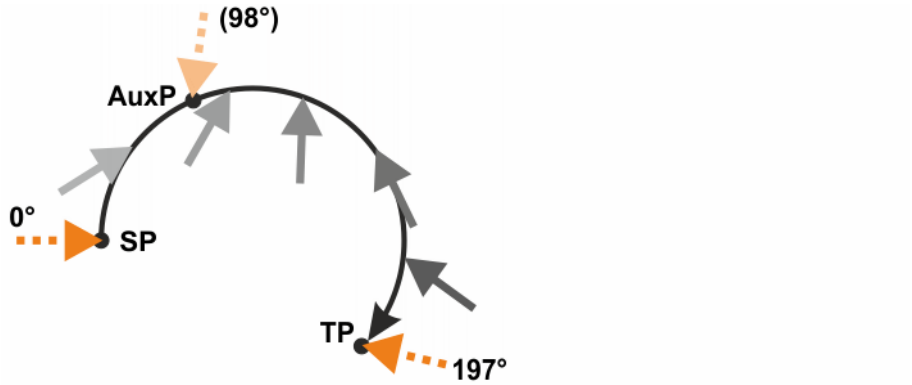


Fig. 9-34: #IGNORE

#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.

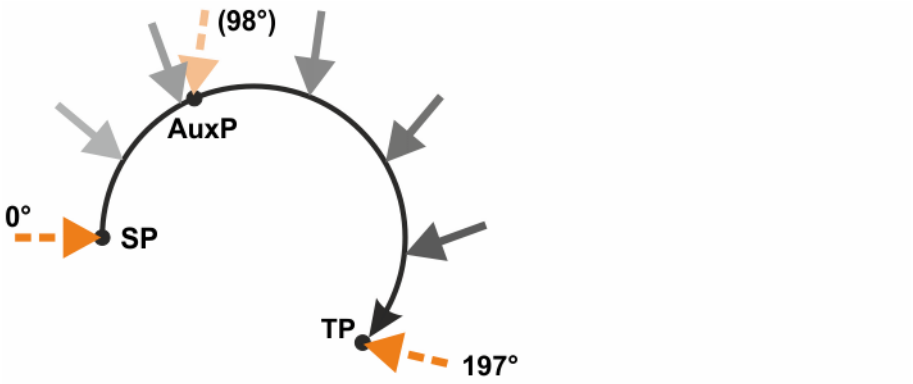


Fig. 9-35: #CONSIDER

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.

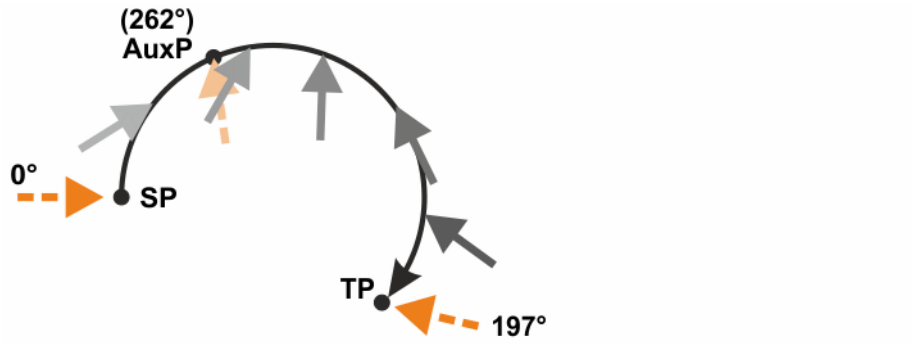


Fig. 9-36: #CONSIDER, additional example

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**.

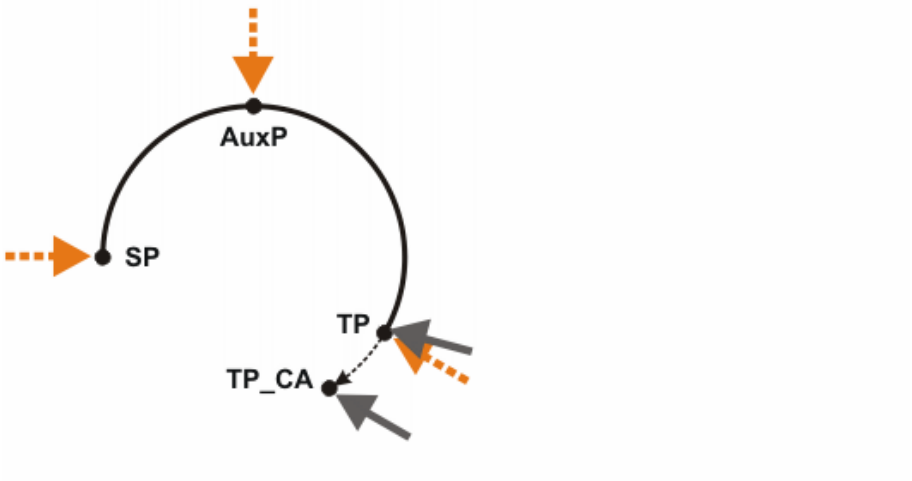


Fig. 9-37: #INTERPOLATE

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.

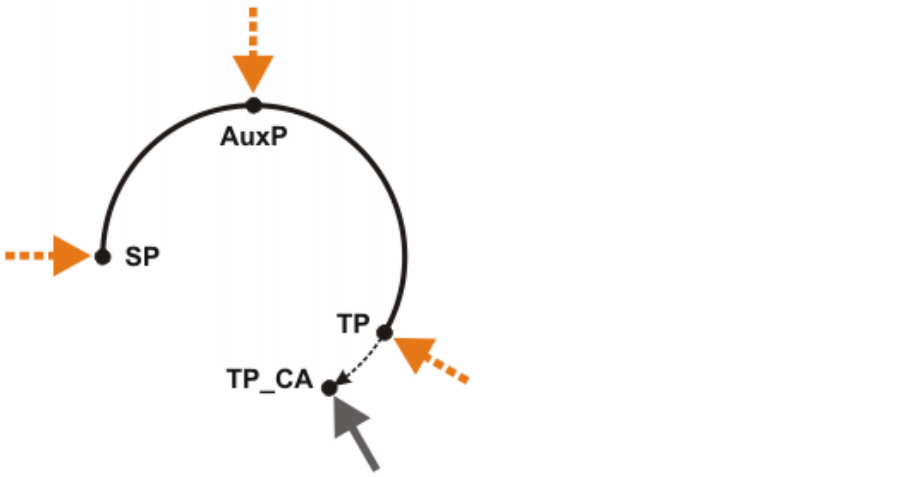


Fig. 9-38: #EXTRAPOLATE

9.9 Circular angle

In KRL, 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 individual motion.

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.

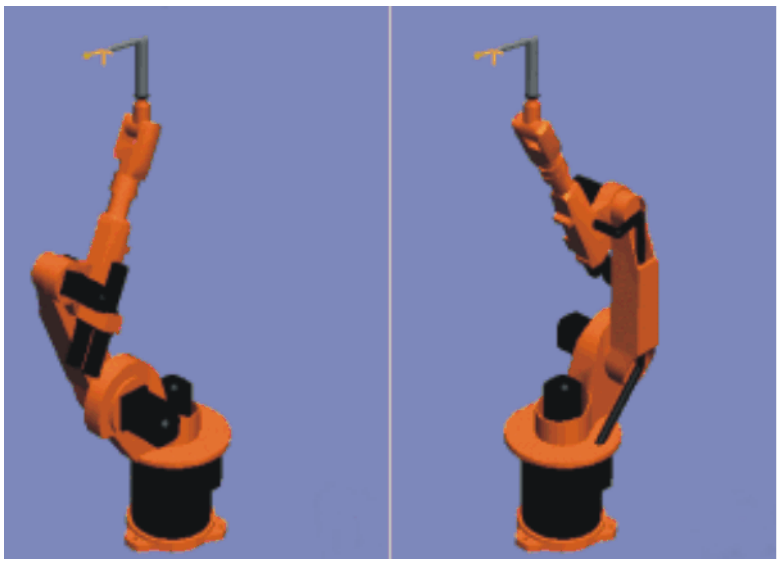


Fig. 9-39: Example: Same TCP position, different axis position

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, INT S, T, REAL E1, E2,
E3, E4, E5, E6
```

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.

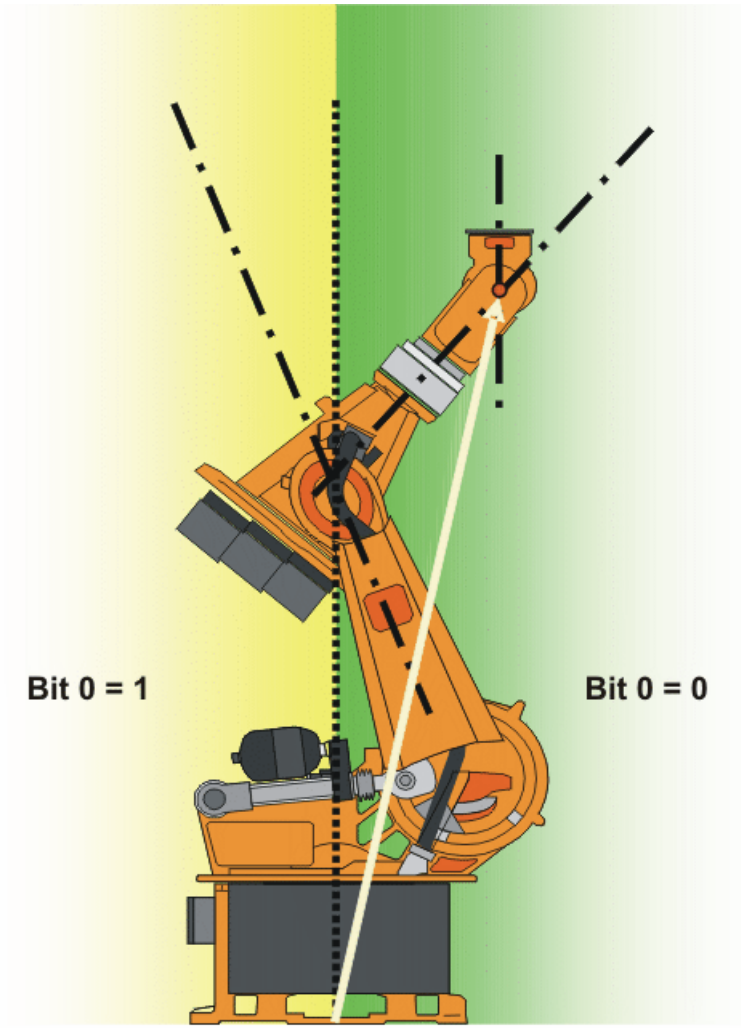


Fig. 9-40: Example: The intersection of the wrist axes (red dot) is in the basic area.

Bit 1

Bit 1 specifies the position of axis A3.

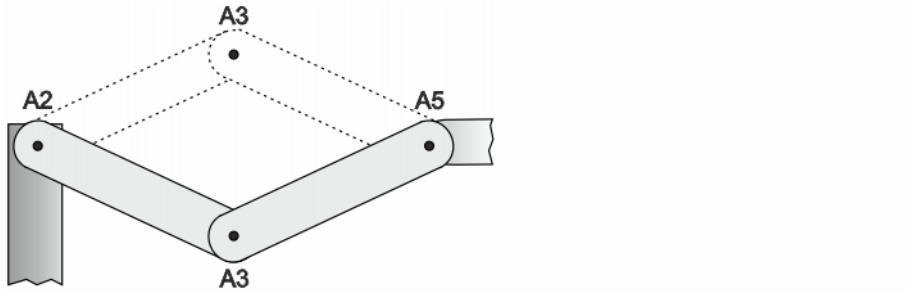


Fig. 9-41: Bit 1 – position of A3

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.

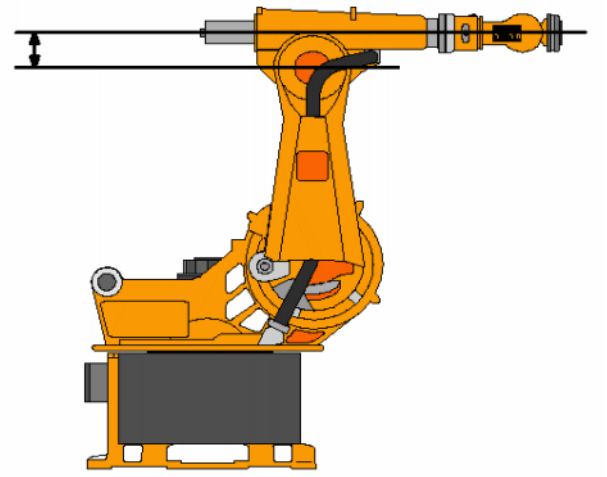


Fig. 9-42: Offset between A3 and A4: example KR 30

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.

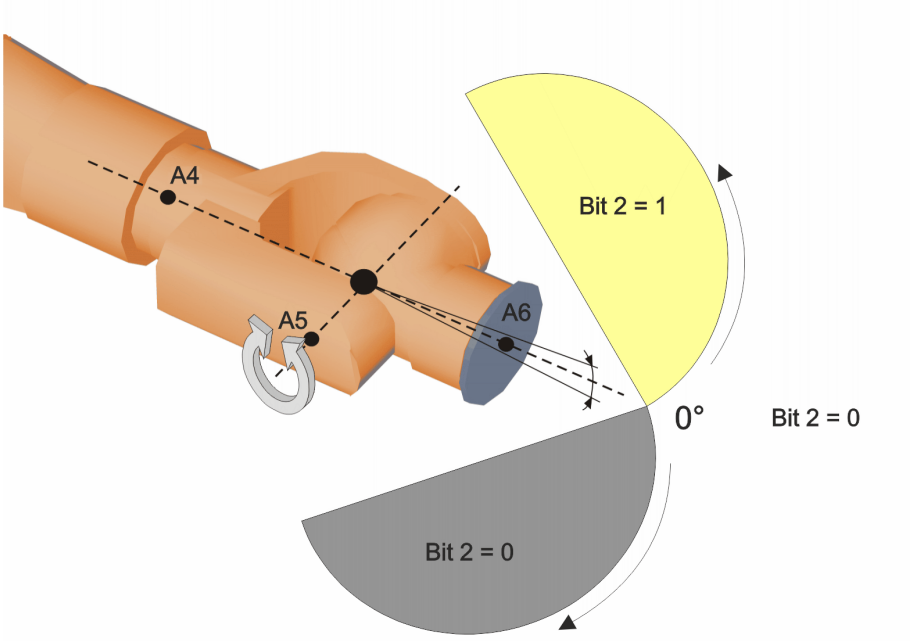


Fig. 9-43: Bit 2

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

Damage to the energy supply system due to unfavorable motion sequence

If motions place excessive tension or pressure on the energy supply system, damage to property may result. The energy supply system, for example, can become wound up and then tear off.

- When programming and testing, ensure that the energy supply system is not damaged during the motion sequence.

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.



It is also possible to insert and change inline forms in KRL programs using WorkVisual.
Further information about editing programs in the KRL editor is contained in the WorkVisual documentation.

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 Inline form PTP: programming a PTP motion

Precondition

- User rights: Function group **Old motion range inline forms**
- Program is selected or open.
- 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.
5. Save instruction with **Cmd OK**.

Inline form

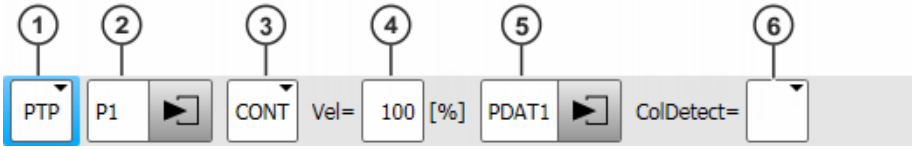


Fig. 10-1: Inline form for PTP motions

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. (>>> *10.3.4 "Option window “Frames”" Page 455*) The arrow can also be used to edit the **Global point** setting.
3	• **CONT**: end point is approximated. • **[blank]**: 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. (>>> *10.3.5 "Option window “Motion parameters” (LIN, CIRC, PTP)" Page 456*)

Item	Description
6	Collision detection for this motion • **[blank]**: This motion has no separate settings for collision detection. If general collision detection is activated, its settings are applicable. If not, collision detection is deactivated. • **CDSet_Set[***No.***]**: Collision detection is activated. The values from data set *No.* are used for detection. If general collision detection is activated at the same time, its settings do not apply to this motion. (>>> *8.14.7 "Activating collision detection for a motion" Page 403*)

10.3.2 Inline form LIN: programming a LIN motion

Precondition

- User rights: Function group **Old motion range inline forms**
- Program is selected or open.
- 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.
5. Save instruction with **Cmd OK**.

Inline form

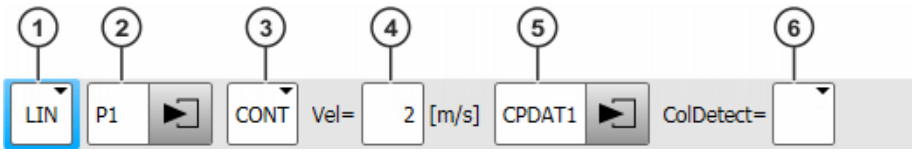


Fig. 10-2: Inline form for LIN motions

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. (>>> *10.3.4 "Option window "Frames"" Page 455*) The arrow can also be used to edit the **Global point** setting.
3	• **CONT**: end point is approximated. • **[blank]**: the motion stops exactly at the end point.
4	Velocity • **0.001 … 2 m/s**

Item	Description
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. (>>> *10.3.5 "Option window “Motion parameters” (LIN, CIRC, PTP)" Page 456*)
6	Collision detection for this motion • **[blank]**: this motion has no separate settings for collision detection. If general collision detection is activated, its settings are applicable. If not, collision detection is deactivated. • **CDSet_Set[***No.***]**: collision detection is activated. The values from data set *No.* are used for detection. If general collision detection is activated at the same time, its settings do not apply to this motion. (>>> *8.14.7 "Activating collision detection for a motion" Page 403*)

10.3.3 Inline form CIRC: programming a CIRC motion

Precondition

- User rights: Function group **Old motion range inline forms**
- Program is selected or open.
- 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.
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**.

Inline form

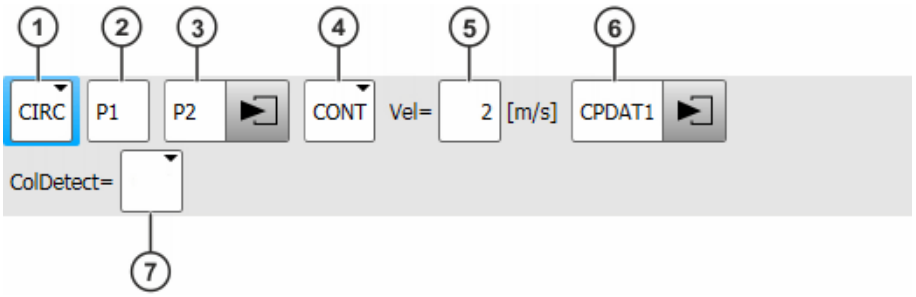


Fig. 10-3: Inline form for CIRC motions

Item	Description
1	Motion type **CIRC**
2	Name of the auxiliary point The system automatically generates a name. The name can be overwritten.
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. (>>> *10.3.4 "Option window “Frames”" Page 455*)
4	• **CONT**: end point is approximated. • **[blank]**: 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. (>>> *10.3.5 "Option window “Motion parameters” (LIN, CIRC, PTP)" Page 456*)
7	Collision detection for this motion • **[blank]**: this motion has no separate settings for collision detection. If general collision detection is activated, its settings are applicable. If not, collision detection is deactivated. • **CDSet_Set[***No.***]**: collision detection is activated. The values from data set *No.* are used for detection. If general collision detection is activated at the same time, its settings do not apply to this motion. (>>> *8.14.7 "Activating collision detection for a motion" Page 403*)

10.3.4 Option window “Frames”

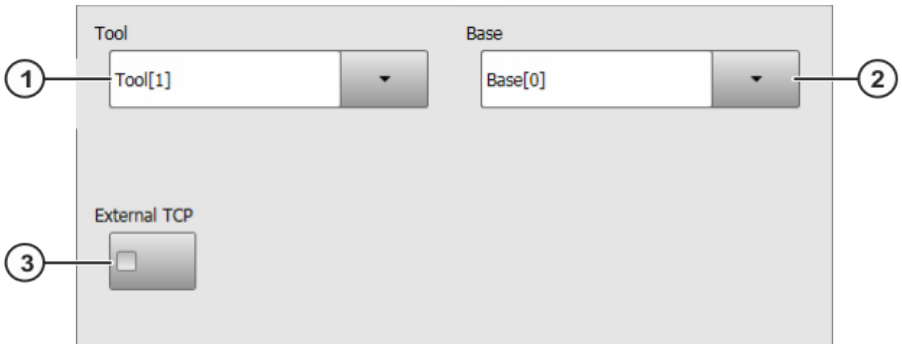


Fig. 10-4: Option window “Frames”

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]
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

10.3.5 Option window “Motion parameters” (LIN, CIRC, PTP)

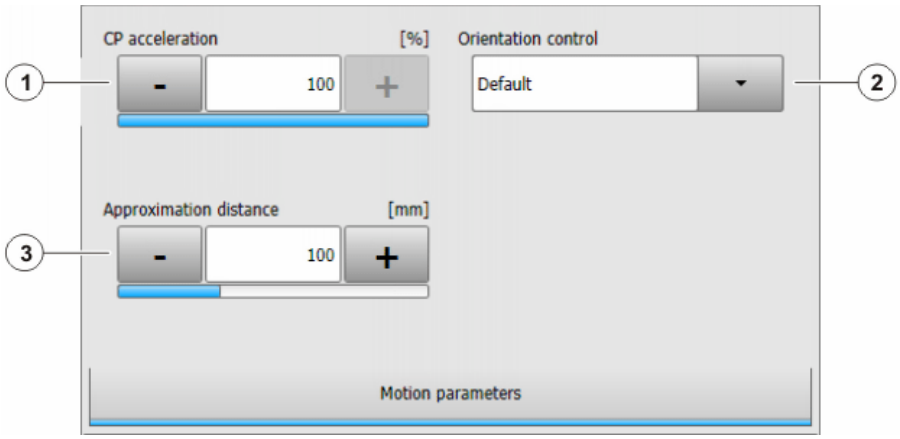


Fig. 10-5: Option window “Motion parameters” (LIN, CIRC, PTP)

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.

Item	Description
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 418*)

10.3.6 Near range approximate positioning

Description

Near range approximate positioning enables approximate positioning of the end point with a definable accuracy.

- The point can be addressed more precisely than with standard approximate positioning.
- The point is nevertheless not addressed with exact positioning. This means time savings compared with exact positioning.

Only the Cartesian position of the robot is evaluated. External axes have no influence on the approximate positioning characteristics.

Triggers behave in the same way as if exact positioning were programmed. In particular, a PATH trigger cannot be shifted beyond the approximate positioning point.

Overview

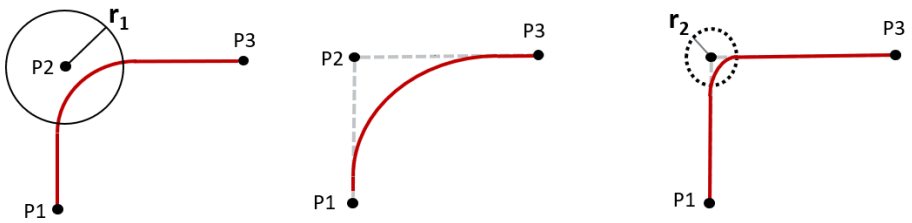


Fig. 10-6: Standard approximate positioning / near range approximate positioning

From left to right:

- Standard approximate positioning
- Standard approximate positioning with distance “0”
- Near range approximate positioning

r = $APO.CDIS; in the case of inline programming, the value of the parameter **Approximation distance**

Programming

	Inline form	KRL
Activation required?	Near range approximate positioning must first be activated via an inline form. (>>> *10.3.6.1 "Activating near range approximate positioning for inline forms" Page 458*)	- - - (no activation required)
Programmable for …	• PTP, LIN, CIRC (>>> *10.3.6.2 "Programming near range approximate positioning via an inline form" Page 459*)	• PTP, LIN, CIRC • PTP_REL, LIN_REL, CIRC_REL For near range approximate positioning, set the parameter **C_APX**. (>>> *11.5.7 "Approximation parameters for PTP, LIN, CIRC and …_REL" Page 562*)
Set accuracy	In the **Motion parameters** option window belonging to the inline form, via **Approximation distance**	Via $APO.CDIS

10.3.6.1 Activating near range approximate positioning for inline forms

Description

The entry **NEAR** in the inline form must first be activated before it is available for near range approximate positioning.

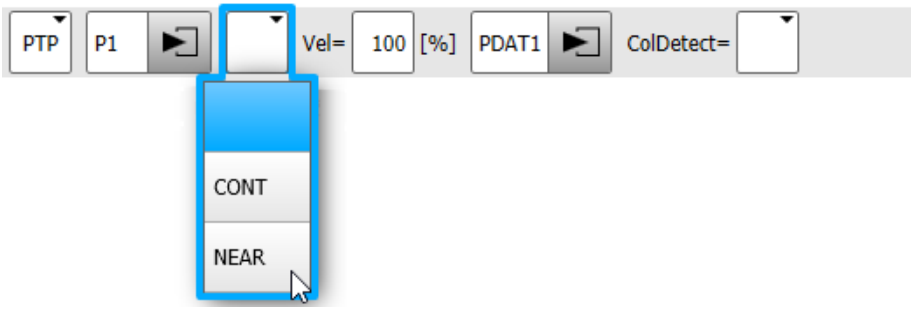


Fig. 10-7: Inline form with NEAR

If **NEAR** is set in the inline form, the default value for **Approximation distance** in the option window **Motion parameters** is 2 mm. (Not 500 mm as with **CONT**.)

Instead of “2 mm”, a different value can be stored as the default value.

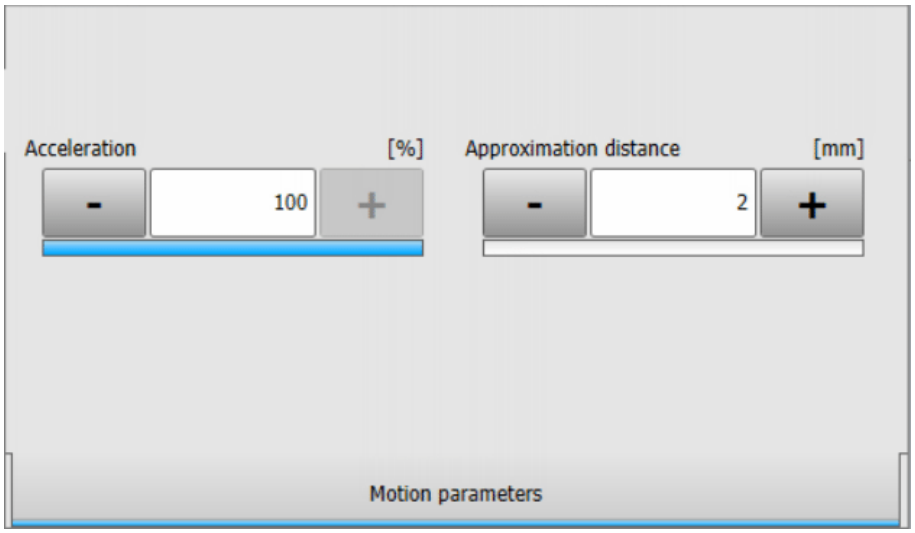


Fig. 10-8: “Approximation distance” after selection of NEAR in the inline form

Precondition

- User group “Expert”
- Operating mode T1 or T2.
- No program is selected.

Procedure

1. Open the file **SmartHMI.User.config** in C:\KRC\USER.
2. To activate approximate positioning, set the following entry to TRUE:
   `KukaRoboter.Techhandler.MovementBasic.Movement-Base.UseNearPointApproximation Value="false"`
3. To change the default value of 2.0 mm for **Approximation distance**, modify the following entry:
   `KukaRoboter.Techhandler.MovementBasic.Movement-Base.APO_DIST_ForNearPointApproximation Value="2.0"`
   Recommended range: 1.0 … 3.0.
4. To apply the changes:
   a. Close the file. Respond to the request for confirmation asking whether the changes should be saved by pressing **Yes**.
   b. Carry out a cold restart. (**Reload files** not necessary.)

10.3.6.2 Programming near range approximate positioning via an inline form

Precondition

- User rights: Function group **Old motion range inline forms**
- Near range approximate positioning is active.
  (>>> *10.3.6.1 "Activating near range approximate positioning for inline forms" Page 458*)
- Program is selected or open.
- T1 mode

Procedure

1. Set **NEAR** in the inline form.

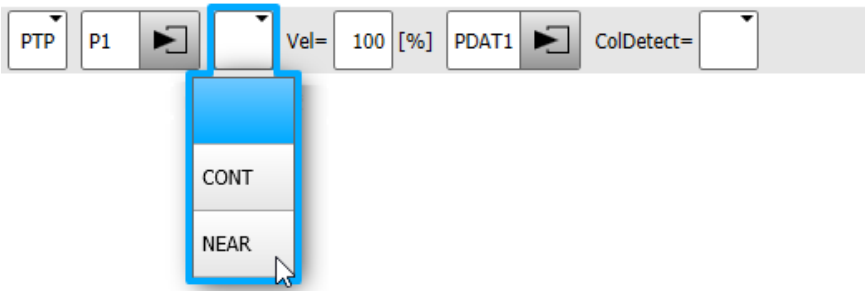


Fig. 10-9: Inline form with NEAR

2. Touch the right arrow next to PDAT[…] or CPDAT[…]. The option window **Motion parameters** is opened.
3. Set the desired accuracy in the **Approximation distance** box.
   - Range of values: 0.00 … 500.00 mm
   - Recommended setting: 1.00 … 3.00 mm.



After the change, the TCP generally moves even further towards the end point in the following motion block. It is therefore advisable to program a higher value than the desired accuracy if motion is to be as time-optimized as possible.

4. Save the inline form with **Cmd OK**.

10.4 Inline form HOP: programming a combined motion

Description

A HOP motion is a motion that bridges a defined height. It is a combination of several motions that do not have to be taught separately, however. Only the end point has to be taught.

- HOP triggers an advance run stop. The end point cannot be approximated.
- The start and end points of a HOP motion can also be located on a conveyor.

HOP motions are particularly suitable for robots with fewer than 6 axes.



HOP motions can also be used for 6-axis robots. For this, it is advisable to switch the 6-axis robot to palletizing mode. This optimizes the motion safety.
(>>> *5.15 "Activating palletizing mode" Page 160*)

Simplified depiction

A HOP motion consists of the following individual motions:

1. Vertically upwards from the start point
2. Horizontally to the position directly over the end point
3. Vertically downwards to the end point

The “corners” can be approximated.

Detailed depiction

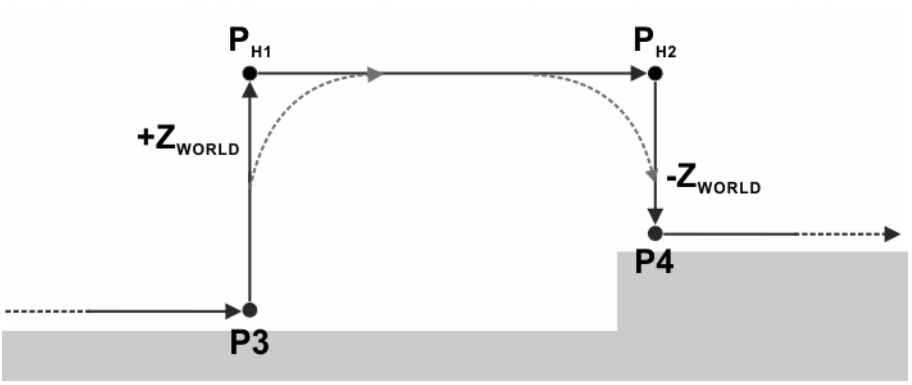


Fig. 10-10: Example: HOP motion from P3 to P4

1. Linear motion from start point (in example **P3**) in direction **+Z_{WORLD}** up to height H1
2. Fastest possible motion to the point located at height H2 above the end point. The value of **Z_{WORLD}** remains the same.
   (= “Transfer motion”)
3. Linear motion in direction **-Z_{WORLD}** from height H2 to the end point (in example **P4**)

The motions into and out of the transfer motion can be approximated.

Indirect transfer motion

If the fastest possible transfer motion cannot be executed because the robot itself is in the way, the controller automatically detects this in advance and executes an indirect transfer motion instead.

(>>> *10.4.1 "Indirect transfer motion" Page 463*)

Precondition

- User rights: Function group **Old motion range inline forms**
- Program is selected or open.
- 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** > **HOP**.
4. If the motion parameters (height, approximation distance, velocity) for the start and target positions are to be different:
   Select **Add target parameters** in the button bar at the bottom.
5. Set the parameters in the inline form.
6. Press **Cmd OK** to save the instruction.

Inline form

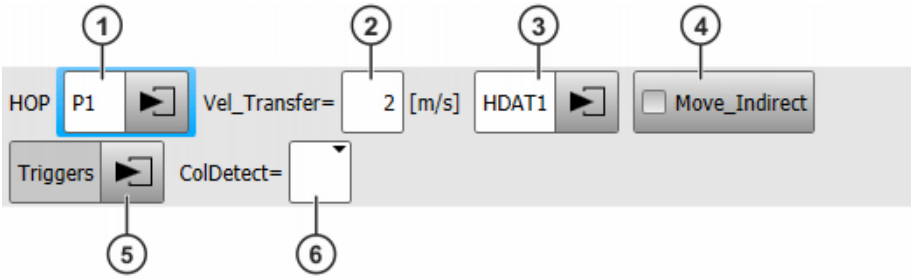


Fig. 10-11: Inline form “HOP”

Item	Description
1	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. (>>> *10.3.4 "Option window “Frames”" Page 455*) The arrow can also be used to edit the **Global point** setting.
2	Velocity for the transfer motion The transfer motion is executed as quickly as possible up to the limit defined here. The velocity that can actually be reached depends on the specific circumstances, and in particular on the robot model. • **0.001 … 15 m/s**
3	(>>> *10.4.2 "Option window for the HOP motion parameters" Page 463*)
4	A check mark next to **Move_Indirect** forces the indirect transfer motion, i.e. the indirect transfer motion is always executed, even if the robot is not standing in its own way. In some cases, this prevents swivel motions that can occur in the “normal” transfer motion. Swivel motions can occur, for example, due to passing through singularity positions.
5	The box is not available as standard. To add it, select **Trigger actions** > **Add trigger command** in the button bar. This creates 1 trigger at the same time. To create further triggers (maximum of 3 in total), select **Trigger actions** > **Add trigger command** again. (>>> *10.4.3 "Option window for HOP Trigger" Page 465*)
6	Collision detection for this motion • **[blank]**: This motion has no separate settings for collision detection. If general collision detection is activated, its settings are applicable. If not, collision detection is deactivated. • **CDSet_Set[***No.***]**: Collision detection is activated. The values from data set *No.* are used for detection. If general collision detection is activated at the same time, its settings do not apply to this motion. (>>> *8.14.7 "Activating collision detection for a motion" Page 403*)

10.4.1 Indirect transfer motion

If the fastest possible transfer motion cannot be executed because the robot itself is in the way, the controller automatically detects this in advance and executes an indirect transfer motion instead.

The controller calculates an optimal curve on the XY plane (WORLD) for the indirect transfer motion.

- The highest point of the curve is located on the halfway line between the start and end points.
- The curve is not necessarily a true circular path.



Fig. 10-12: Direct path not possible (view of robot from top)

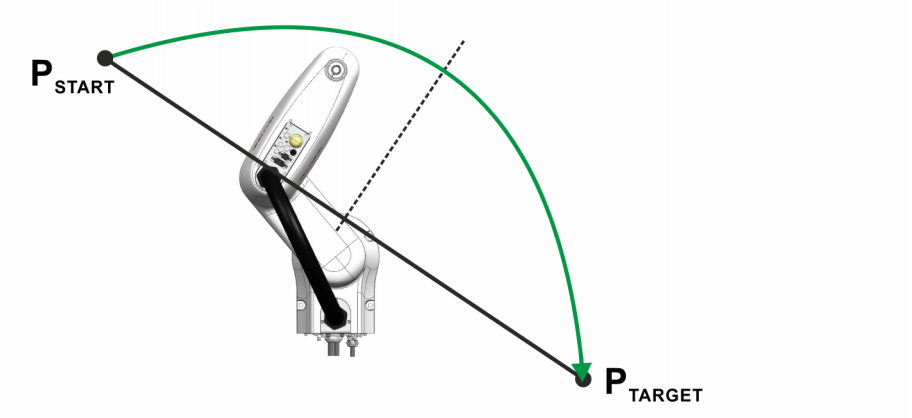


Fig. 10-13: Indirect transfer motion (view of robot from top)

10.4.2 Option window for the HOP motion parameters

As standard, the motion parameters for the start and end positions are identical and the **Motion parameters for the start and target position** option window is displayed.

To define different motion parameters for the two positions:

- After opening the inline form, select the **Add target parameters** button.

Separate option windows are now displayed for the start and end positions. **Delete target parameters** can be used to undo the action.

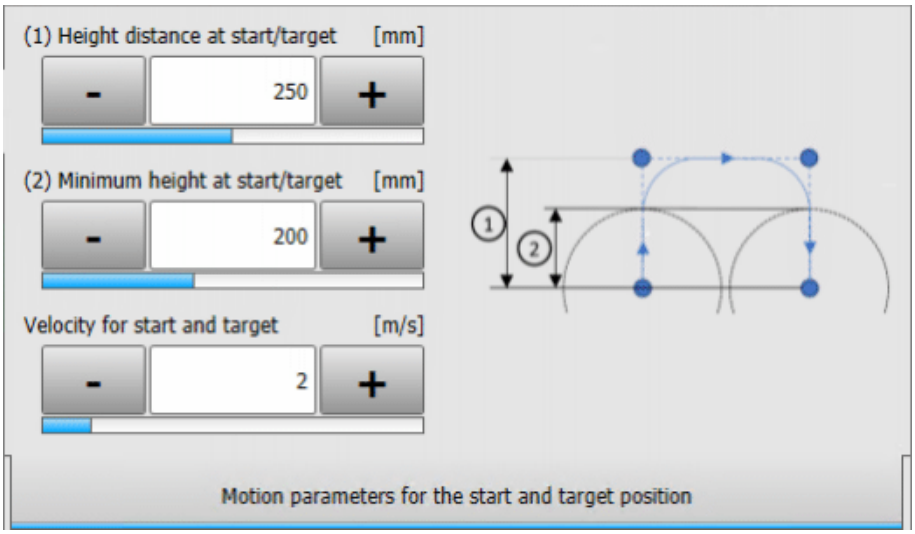


Fig. 10-14: Motion parameters for the start and target position

Parameter	Description
Note: Height from the point in the direction **+Z_{WORLD}**	
(1) Height distance at start/target	• Height above the start point at which the transfer motion is to start • And simultaneously height above the end point at which the transfer motion is to end Height value without consideration of possible approximate positioning in each case • **0 … 500 mm**
(2) Minimum height at start/target	• Height above the start point at which the approximate positioning into the transfer motion is to start • And simultaneously height above the end point at which the approximate positioning out of the transfer motion is to end If no approximate positioning is desired, the same value must be entered as under **(1) Height distance at start/target**. • **0 … 500 mm**
Velocity for start and target	• Velocity upwards from the start point to the start of the transfer motion • And simultaneously velocity downwards from the end of the transfer motion to the end point The velocity that can actually be reached depends on the specific circumstances, and in particular on the robot model. • **0 … 15 m/s**

10.4.3 Option window for HOP Trigger

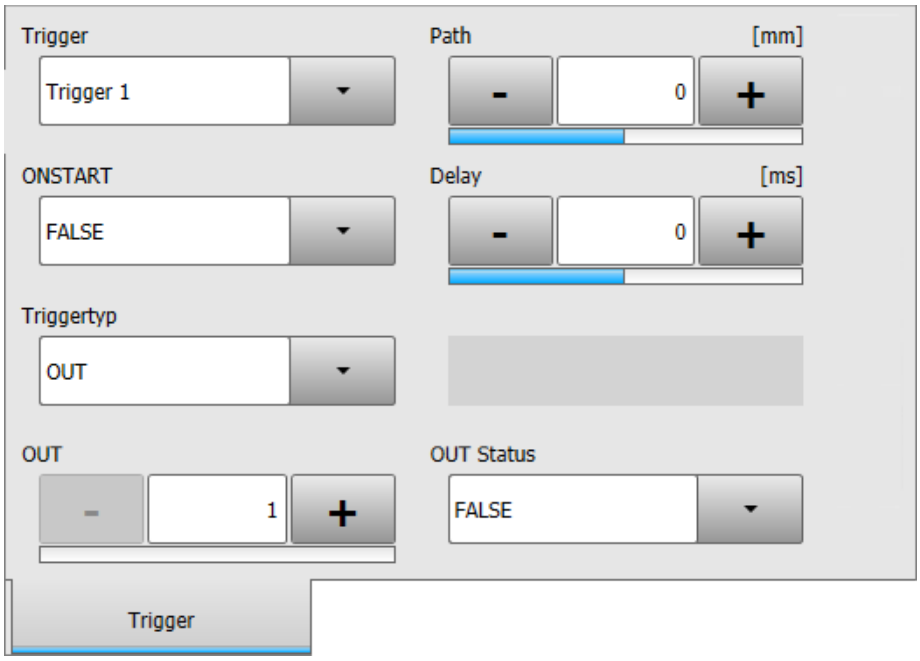


Fig. 10-15: Trigger, example: trigger type OUT

General settings (>>> *Fig. 10-15*):

Box	Description
Trigger	Trigger with number; maximum 3 triggers
Path	Shift in space, starting from the reference point. If no shift in space is desired, enter “0”. • -2,000 … +2,000 mm Negative value: Offset towards the start of the motion Positive value: Offset towards the end of the motion (>>> *"Max. offset" Page 639*)
ONSTART	Reference point of the trigger • FALSE: End point of the HOP motion • TRUE: Point $\mathbf{P_{H2}}$ (>>> *Fig. 10-10*) If TRUE, **Path** automatically becomes 0. (>>> *11.12.2.1 "Reference point for approximate positioning – overview" Page 642*)
Delay	Shift in time relative to the PATH value. If no shift in time is desired, enter “0”. • -1,000 … +1,000 ms Negative value: Offset towards the start of the motion Positive value: Trigger is switched after **Delay** has elapsed. (>>> *"Max. offset" Page 639*)
Trigger type	OUT, PULSE, ASSIGNMENT or CALL

Trigger types **OUT** (>>> *Fig. 10-15*) and **PULSE**:

Box	Description
OUT	Number of the digital output • 1 … 8192
OUT status	State to which the output is switched • TRUE: “High” level • FALSE: “Low” level
Pulse time	Only for trigger type PULSE: Length of the pulse • -1,000 … +1,000 ms

Trigger type **ASSIGNMENT** (not illustrated):

Box	Description
Assignment variable	Variable to which a value is to be assigned **Note:** Runtime variables cannot be used.
Assigned value	Value to be assigned to the variable

Trigger type **CALL** (not illustrated):

Box	Description
Method call	Name of the subprogram to be called
Priority	Priority of the trigger 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.

10.5 Programming spline motions

10.5.1 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.

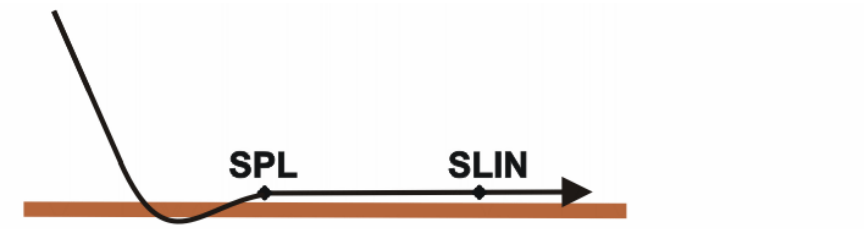


Fig. 10-16: Collision with work surface

In order to avoid a collision, observe the recommendations for the SLIN-SPL-SLIN transition.

(>>> *9.7.6.1 "SLIN-SPL-SLIN transition" Page 435*)

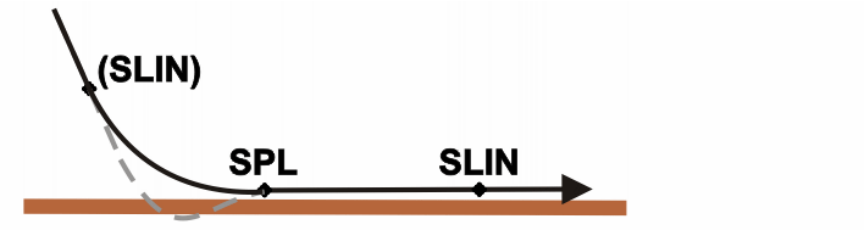


Fig. 10-17: Avoiding a collision with the work surface

- 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.

10.5.2 Programming a spline block

10.5.2.1 Inline form “Spline”: programming a CP 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 option 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**
- Program is selected or open.
- 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** > **SPLINE block**.
3. Set the parameters in the inline form.
4. Press **Cmd OK**.
5. Press **Open/close fold**. Spline segments can now be inserted into the block.

Inline form

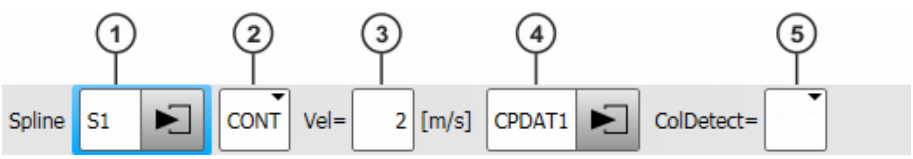


Fig. 10-18: Inline form “Spline”

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. (>>> *10.3.4 "Option window “Frames”" Page 455*)
2	• **CONT**: end point is approximated. • **[blank]**: 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. (>>> *10.5.2.3 "Option window “Motion parameters” (CP spline block)" Page 471*)
5	Collision detection for the spline block. The setting is valid for the segments for which the **ColDetect** box is not displayed. • **[blank]**: this motion has no separate settings for collision detection. If general collision detection is activated, its settings are applicable. If not, collision detection is deactivated. • **CDSet_Set[***No.***]**: collision detection is activated. The values from data set *No.* are used for detection. If general collision detection is activated at the same time, its settings do not apply to this motion. (>>> *8.14.7 "Activating collision detection for a motion" Page 403*)

10.5.2.2 Inline form “PTP spline”: programming a PTP 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 option 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**
- Program is selected or open.
- 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** > **PTP SPLINE block**.
3. Set the parameters in the inline form.
4. Press **Cmd OK**.
5. Press **Open/close fold**. Spline segments can now be inserted into the block.

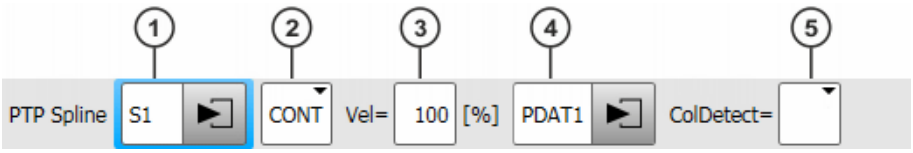


Fig. 10-19: Inline form “PTP spline”

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. (>>> *10.3.4 "Option window “Frames”" Page 455*)
2	• **CONT**: end point is approximated. • **[blank]**: 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. (>>> *10.5.2.4 "Option window “Motion parameters” (PTP spline block)" Page 472*)

Item	Description
5	Collision detection for the spline block. The setting is valid for the segments for which the **ColDetect** box is not displayed. • **[blank]**: this motion has no separate settings for collision detection. If general collision detection is activated, its settings are applicable. If not, collision detection is deactivated. • **CDSet_Set[***No.***]**: collision detection is activated. The values from data set *No.* are used for detection. If general collision detection is activated at the same time, its settings do not apply to this motion. (>>> *8.14.7 "Activating collision detection for a motion" Page 403*)

10.5.2.3 Option window “Motion parameters” (CP spline block)

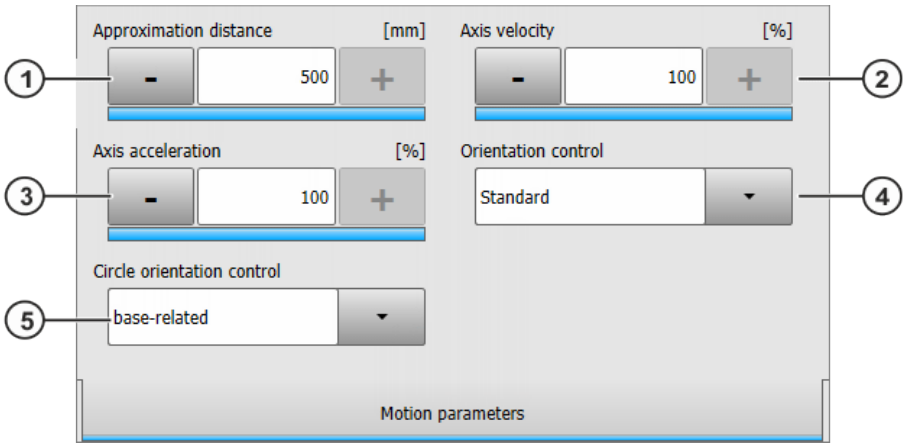


Fig. 10-20: Option window “Motion parameters” (CP spline block)

Item	Description
1	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. • **0 … 500 mm**
2	Axis velocity. The value refers to the maximum value specified in the machine data. • **1 … 100%**
3	Axis 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).

10.5.2.4 Option window “Motion parameters” (PTP spline block)

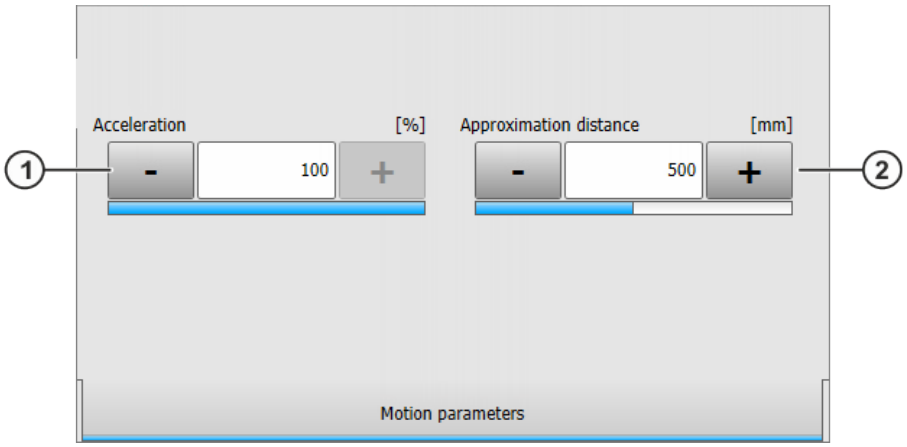


Fig. 10-21: Option window “Motion parameters” (PTP spline block)

Item	Description
1	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. • **0 … 500 mm**

10.5.3 Programming segments for a spline block

10.5.3.1 Inline form “SPL”: Programming the SPL segment

Precondition

- User rights: Function group **New motion range inline forms**
- Program is selected or open.
- 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**.
4. Set the parameters in the inline form.
5. Press **Cmd OK**.

Inline form

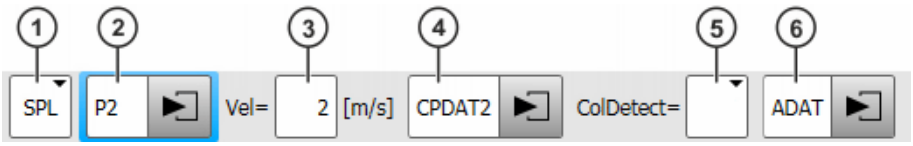


Fig. 10-22: Inline form “SPL segment”

As standard, 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**
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. (>>> *10.5.3.5 "Option window “Motion parameters” (CP spline segment)" Page 479*)
5	Collision detection for this segment • **ColDetect** box is hidden: The setting that also applies to the spline block is valid for this segment. • **OFF**: collision detection is deactivated. • **CDSet_Set[***No.***]**: collision detection is activated. The values from data set *No.* are used for detection. (>>> *8.14.7 "Activating collision detection for a motion" Page 403*)

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. (>>> *10.5.5.1 "Programming a trigger at the spline segment or individual spline instruction" Page 488*) (>>> *10.5.6.1 "Programming a conditional stop at the spline segment or individual spline instruction" Page 493*) (>>> *10.5.7 "Programming a constant velocity range for a CP spline block" Page 498*)

10.5.3.2 Inline form “SLIN”: Programming the SLIN segment

Precondition

- User rights: Function group **New motion range inline forms**
- Program is selected or open.
- 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** > **SLIN**.
4. Set the parameters in the inline form.
5. Press **Cmd OK**.

Inline form

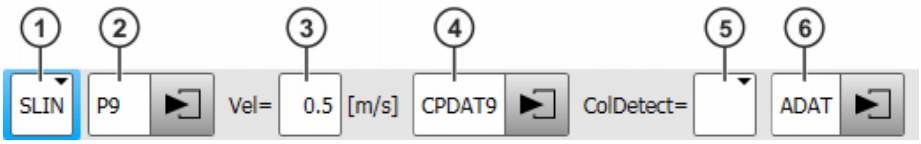


Fig. 10-23: Inline form “SLIN segment”

As standard, 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 **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.

Item	Description
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. (>>> *10.5.3.5 "Option window “Motion parameters” (CP spline segment)" Page 479*)
5	Collision detection for this segment • **ColDetect** box is hidden: The setting that also applies to the spline block is valid for this segment. • **OFF**: collision detection is deactivated. • **CDSet_Set[***No.***]**: collision detection is activated. The values from data set *No.* are used for detection. (>>> *8.14.7 "Activating collision detection for a motion" Page 403*)
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. (>>> *10.5.5.1 "Programming a trigger at the spline segment or individual spline instruction" Page 488*) (>>> *10.5.6.1 "Programming a conditional stop at the spline segment or individual spline instruction" Page 493*) (>>> *10.5.7 "Programming a constant velocity range for a CP spline block" Page 498*)

10.5.3.3 Inline form “SCIRC”: Programming the SCIRC segment

Precondition

- User rights: Function group **New motion range inline forms**
- Program is selected or open.
- 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**.

Inline form

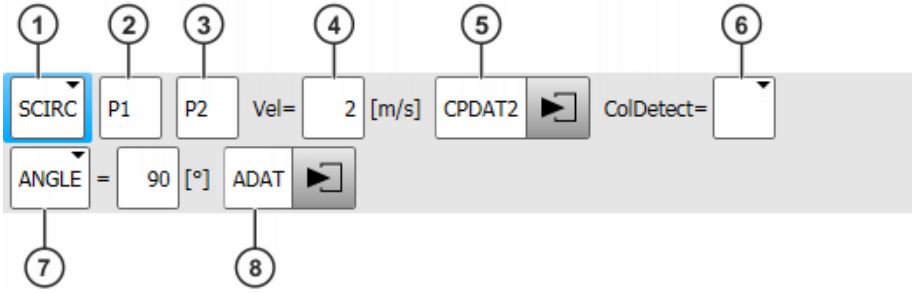


Fig. 10-24: Inline form “SCIRC spline segment”

As standard, 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.
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. (>>> *10.5.3.5 "Option window “Motion parameters” (CP spline segment)" Page 479*)
6	Collision detection for this segment • **ColDetect** box is hidden: the setting that also applies to the spline block is valid for this segment. • **OFF**: collision detection is deactivated. • **CDSet_Set[***No.***]**: collision detection is activated. The values from data set *No.* are used for detection. (>>> *8.14.7 "Activating collision detection for a motion" Page 403*)

Item	Description
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. (>>> *10.5.5.1 "Programming a trigger at the spline segment or individual spline instruction" Page 488*) (>>> *10.5.6.1 "Programming a conditional stop at the spline segment or individual spline instruction" Page 493*) (>>> *10.5.7 "Programming a constant velocity range for a CP spline block" Page 498*)

10.5.3.4 Inline form “SPTP”: Programming the SPTP segment

Precondition

- User rights: Function group **New motion range inline forms**
- Program is selected or open.
- 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.
5. Press **Cmd OK**.

Inline form

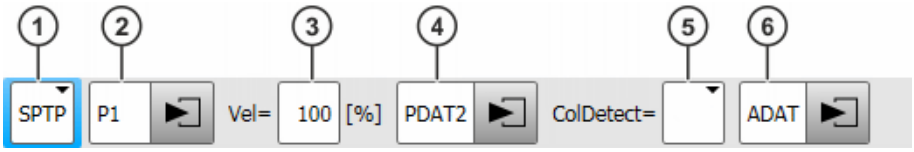


Fig. 10-25: Inline form for SPTP segment

As standard, 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.

Item	Description
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. (>>> *10.5.3.6 "Option window “Motion parameters” (SPTP)" Page 480*)
5	Collision detection for this segment • **ColDetect** box is hidden: The setting that also applies to the spline block is valid for this segment. • **OFF**: collision detection is deactivated. • **CDSet_Set[***No.***]**: collision detection is activated. The values from data set *No.* are used for detection. (>>> *8.14.7 "Activating collision detection for a motion" Page 403*)
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. (>>> *10.5.5.1 "Programming a trigger at the spline segment or individual spline instruction" Page 488*) (>>> *10.5.6.1 "Programming a conditional stop at the spline segment or individual spline instruction" Page 493*)

10.5.3.5 Option window “Motion parameters” (CP spline segment)

Motion parameters

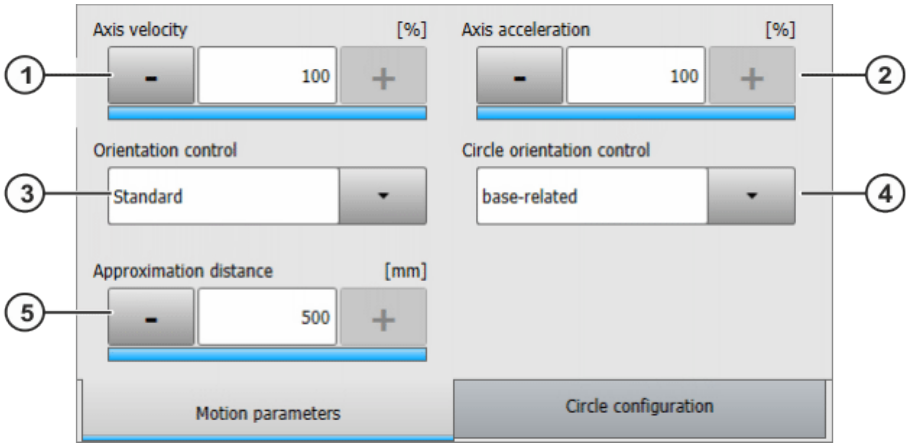


Fig. 10-26: Option window “Motion parameters” (CP spline segment)

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	Orientation control selection.
4	Only in the case of SCIRC segments: orientation control reference system selection.
5	This box is only displayed if **CONT** was selected for the spline block. 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. • **0 … 500 mm**

Circle configuration

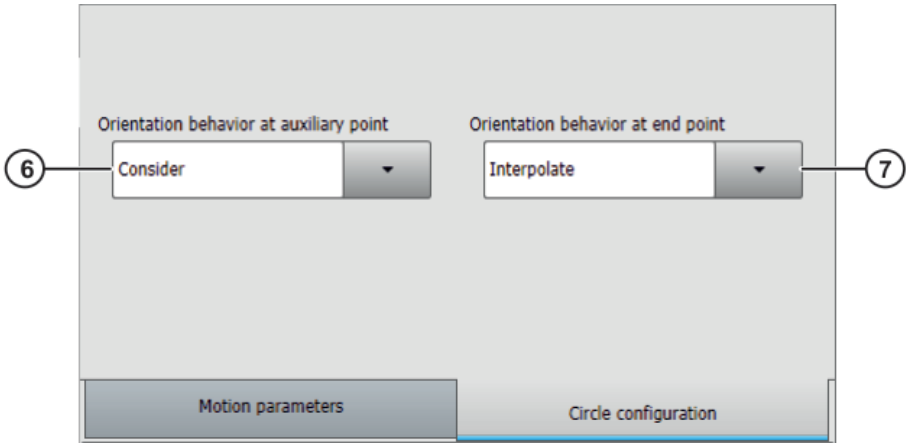


Fig. 10-27: Circle configuration (SCIRC segment)

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.

10.5.3.6 Option window “Motion parameters” (SPTP)

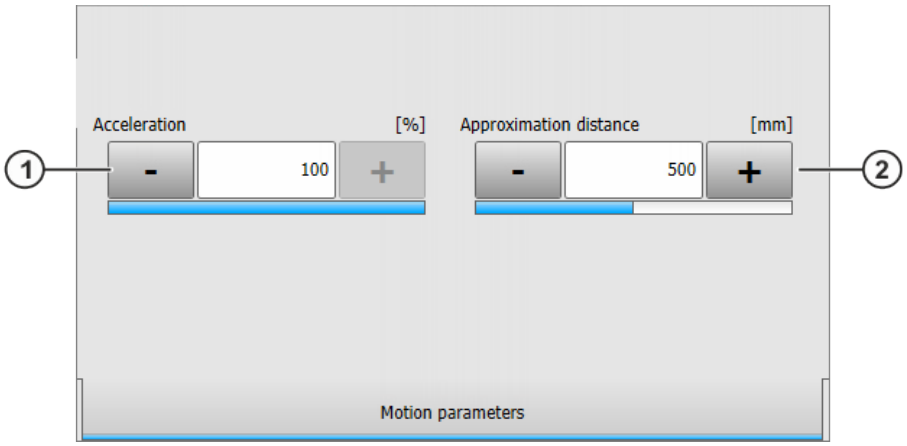


Fig. 10-28: Option window “Motion parameters” (SPTP)

Item	Description
1	Acceleration. The value refers to the maximum value specified in the machine data. • **1 … 100%**
2	In the case of SPTP segments, this box is only displayed if **CONT** was selected in the spline block. 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. • **0 … 500 mm**

10.5.4 Programming individual spline motions

10.5.4.1 Inline form “SLIN”: Programming an individual SLIN motion

Precondition

- User rights: Function group **New motion range inline forms**
- Program is selected or open.
- 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.
5. Press **Cmd OK**.

Inline form

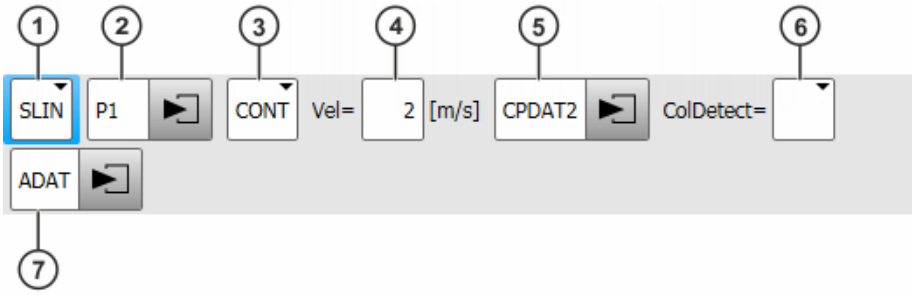


Fig. 10-29: Inline form “SLIN” (individual motion)

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. (>>> *10.3.4 "Option window “Frames”" Page 455*) The arrow can also be used to edit the **Global point** setting.
3	• **CONT**: end point is approximated. • **[blank]**: 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. (>>> *10.5.4.2 "Option window “Motion parameters” (SLIN)" Page 483*)
6	Collision detection for this motion • **[blank]**: this motion has no separate settings for collision detection. If general collision detection is activated, its settings are applicable. If not, collision detection is deactivated. • **CDSet_Set[***No.***]**: Collision detection is activated. The values from data set *No.* are used for detection. If general collision detection is activated at the same time, its settings do not apply to this motion. (>>> *8.14.7 "Activating collision detection for a motion" Page 403*)
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. (>>> *10.5.5.1 "Programming a trigger at the spline segment or individual spline instruction" Page 488*) (>>> *10.5.6.1 "Programming a conditional stop at the spline segment or individual spline instruction" Page 493*)

10.5.4.2 Option window “Motion parameters” (SLIN)

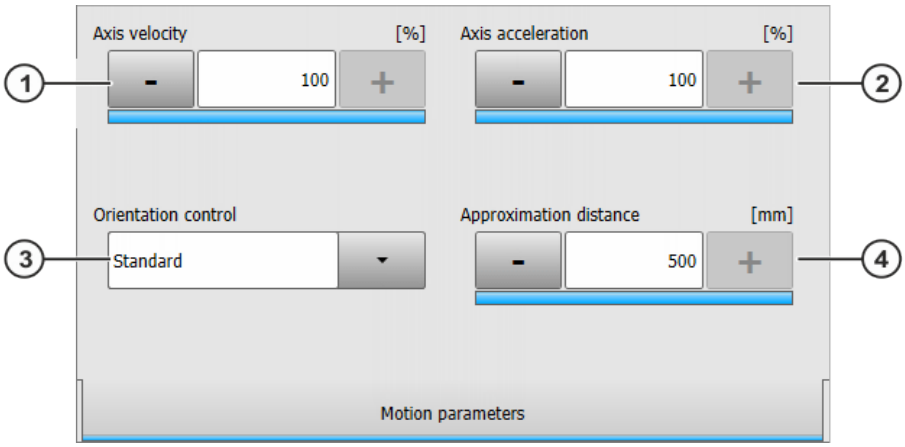


Fig. 10-30: Option window “Motion parameters” (SLIN)

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%**
3	Orientation control selection.
4	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. • **0 … 500 mm**

10.5.4.3 Inline form “SCIRC”: Programming an individual SCIRC motion

Precondition

- User rights: Function group **New motion range inline forms**
- Program is selected or open.
- 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.
5. Press **Teach Aux**.
6. Move the TCP to the end point.
7. Press **Cmd OK**.

Inline form

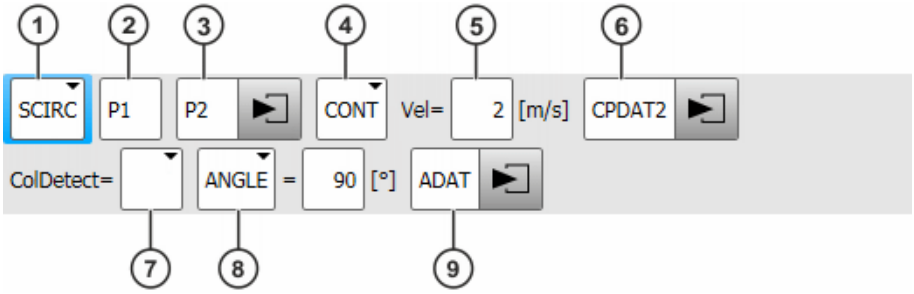


Fig. 10-31: Inline form “SCIRC” (individual motion)

Item	Description
1	Motion type **SCIRC**
2	Point name for the auxiliary point. The system automatically generates a name. The name can be overwritten.
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. (>>> *10.3.4 "Option window “Frames”" Page 455*)
4	• **CONT**: end point is approximated. • **[blank]**: 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. (>>> *10.5.4.4 "Option window “Motion parameters” (SCIRC)" Page 485*)
7	Collision detection for this motion • **[blank]**: This motion has no separate settings for collision detection. If general collision detection is activated, its settings are applicable. If not, collision detection is deactivated. • **CDSet_Set[***No.***]**: Collision detection is activated. The values from data set *No.* are used for detection. If general collision detection is activated at the same time, its settings do not apply to this motion. (>>> *8.14.7 "Activating collision detection for a motion" Page 403*)

Item	Description
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. (>>> *10.5.5.1 "Programming a trigger at the spline segment or individual spline instruction" Page 488*) (>>> *10.5.6.1 "Programming a conditional stop at the spline segment or individual spline instruction" Page 493*)

10.5.4.4 Option window “Motion parameters” (SCIRC)

Motion parameters

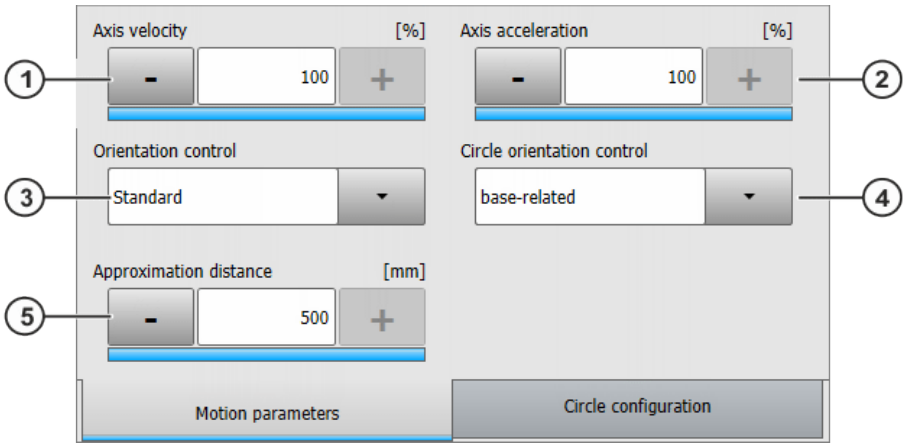


Fig. 10-32: Motion parameters (SCIRC)

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	Orientation control selection.
4	Orientation control reference system selection.

Item	Description
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. • **0 … 500 mm**

Circle configuration

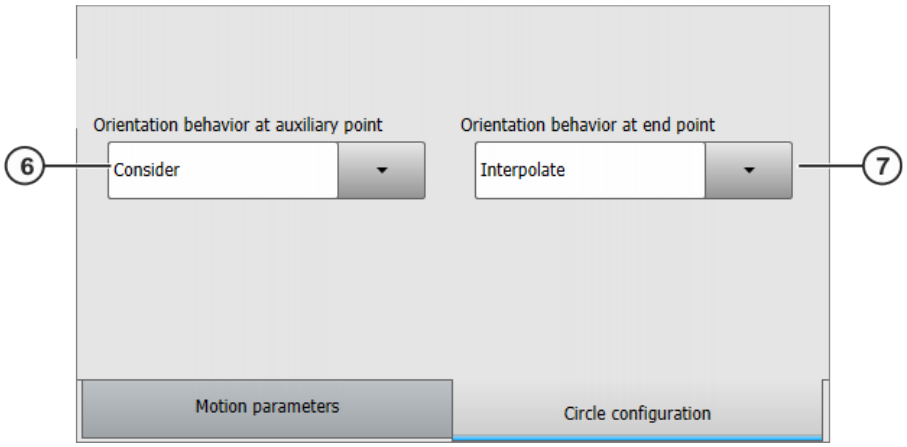


Fig. 10-33: Circle configuration (SCIRC)

Item	Description
6	Selection of orientation behavior at auxiliary point.
7	This box is only displayed if **ANGLE** was selected in the inline form. Selection of orientation behavior at end point.

10.5.4.5 Inline form “SPTP”: Programming an individual SPTP motion

Precondition

- User rights: Function group **New motion range inline forms**
- Program is selected or open.
- 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.
5. Press **Cmd OK**.

Inline form

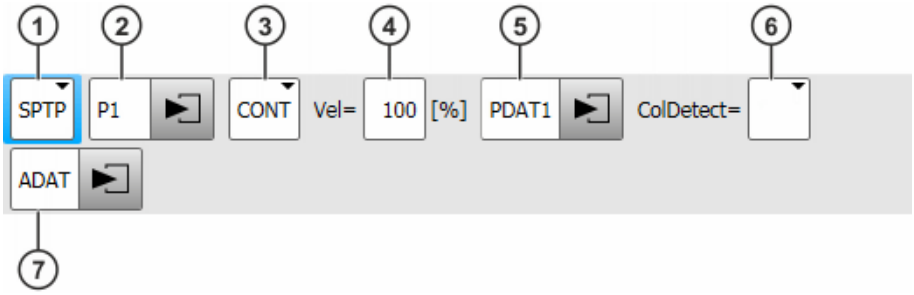


Fig. 10-34: Inline form “SPTP” (individual motion)

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. (>>> *10.3.4 "Option window “Frames”" Page 455*) The arrow can also be used to edit the **Global point** setting.
3	• **CONT**: end point is approximated. • **[blank]**: 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. (>>> *10.5.3.6 "Option window “Motion parameters” (SPTP)" Page 480*)
6	Collision detection for this motion • **[blank]**: this motion has no separate settings for collision detection. If general collision detection is activated, its settings are applicable. If not, collision detection is deactivated. • **CDSet_Set[***No.***]**: Collision detection is activated. The values from data set *No.* are used for detection. If general collision detection is activated at the same time, its settings do not apply to this motion. (>>> *8.14.7 "Activating collision detection for a motion" Page 403*)
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. (>>> *10.5.5.1 "Programming a trigger at the spline segment or individual spline instruction" Page 488*) (>>> *10.5.6.1 "Programming a conditional stop at the spline segment or individual spline instruction" Page 493*)

10.5.5 Programming a trigger for spline

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.

Programming

- At the spline segment or individual spline instruction:
  (>>> *10.5.5.1 "Programming a trigger at the spline segment or individual spline instruction" Page 488*)
- In the spline block (via a separate inline form):
  (>>> *10.5.5.2 "Inline form TRIGGER WHEN PATH: programming a trigger in the spline block" Page 490*)

10.5.5.1 Programming a trigger at the spline segment or individual spline instruction

Precondition

- User rights: Function group **New motion range inline forms**
- Program is selected or open.
- T1 mode

Procedure

1. Open the inline form of the spline segment or individual instruction.
2. To display **ADAT** in the inline form (if not already displayed):
   - For segments: Select **Switch parameter** > **Spline logic**.
   - For individual instructions: Press the **Spline logic** button.
3. Touch the arrow next to **ADAT**: the option window for the logic parameters opens.
4. Select the **Trigger** tab.
   - To add a trigger:
     Select the **Spline actions** button, then **Add trigger command**.
   - To edit an existing trigger: open the box and select the trigger.

   The box is always displayed empty when the tab is opened, even if triggers are already available. These only become visible on opening the box.
5. Make the required settings.
6. Save the settings with **Cmd OK** and close the inline form.

“Trigger” tab

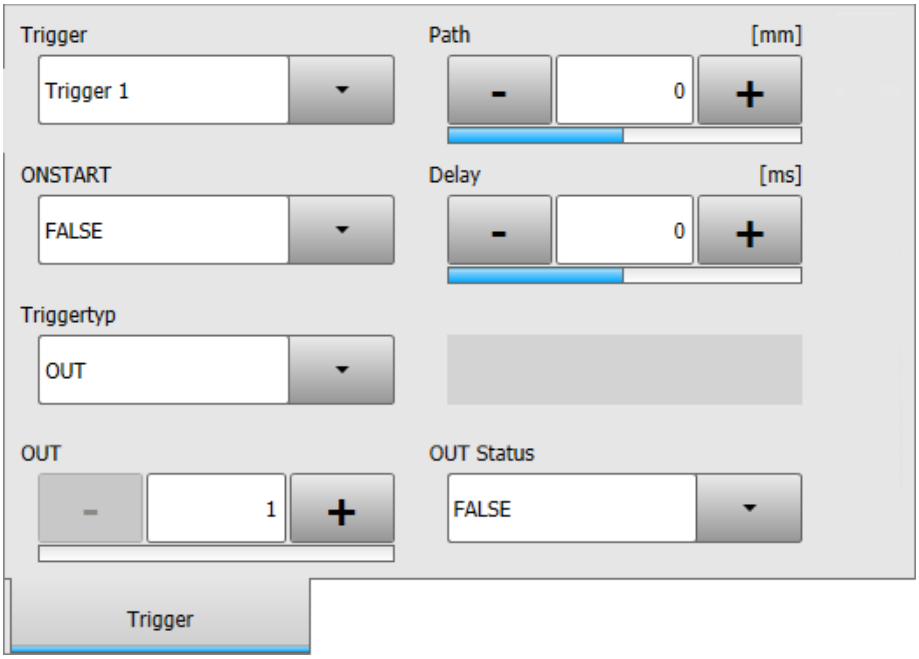


Fig. 10-35: Trigger, example: trigger type OUT

General settings (>>> *Fig. 10-35*):

Box	Description
Trigger	Trigger with number; maximum 8 triggers **Note:** The box is always displayed empty when the tab is opened, even if triggers are already available. These only become visible on opening the box.
Path	Shift in space, starting from the reference point. If no shift in space is desired, enter “0”. • -2,000 … +2,000 mm Negative value: offset towards the start of the motion Positive value: offset towards the end of the motion (>>> *"Max. offset" Page 639*) The shift in space can also be taught instead of being defined here numerically. In this case, the box **ONSTART** is automatically set to **FALSE**. (>>> *10.5.8 "Teaching “Path” for trigger, conditional stop or constant velocity range" Page 501*)
ONSTART	Reference point of the trigger • TRUE: Start point • FALSE: End point (>>> *11.12.2.1 "Reference point for approximate positioning – overview" Page 642*)

Box	Description
Delay	Shift in time relative to the PATH value. If no shift in time is desired, enter “0”. • -1,000 … +1,000 ms Negative value: offset towards the start of the motion Positive value: trigger is switched after **Delay** has elapsed. (>>> *"Max. offset" Page 639*)
Trigger type	OUT, PULSE, ASSIGNMENT or CALL

Trigger types **OUT** (>>> *Fig. 10-35*) and **PULSE**:

Box	Description
OUT	Number of the digital output • 1 … 8192
OUT status	State to which the output is switched • TRUE: “High” level • FALSE: “Low” level
Pulse time	Only for trigger type PULSE: Length of the pulse • -1,000 … +1,000 ms

Trigger type **ASSIGNMENT** (not illustrated):

Box	Description
Assignment variable	Variable to which a value is to be assigned **Note:** Runtime variables cannot be used.
Assigned value	Value to be assigned to the variable

Trigger type **CALL** (not illustrated):

Box	Description
Method call	Name of the subprogram to be called
Priority	Priority of the trigger 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.

10.5.5.2 Inline form TRIGGER WHEN PATH: programming a trigger in the spline block

Precondition

- User rights: Function group **New motion range inline forms**
- Program is selected or open.
- T1 mode
- The spline block fold is open.

Procedure

1. Position the cursor in the line after which the trigger is to be inserted in the spline block.
2. Select the menu sequence **Commands** > **Motion parameters** > **Spline trigger**.
3. Set the parameters in the inline form.
4. Press **Cmd OK**.

Inline form TRIGGER WHEN PATH

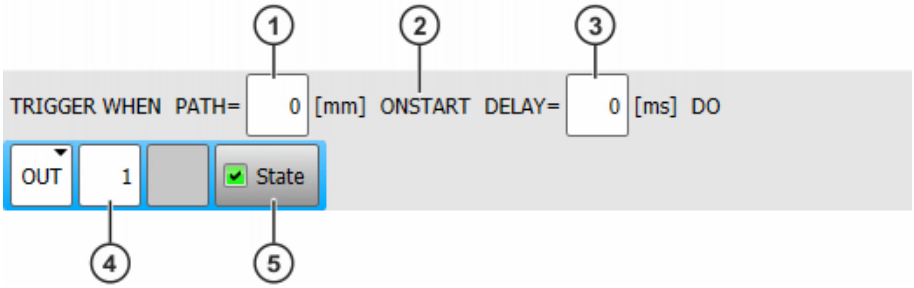


Fig. 10-36: Inline form TRIGGER WHEN PATH, type OUT

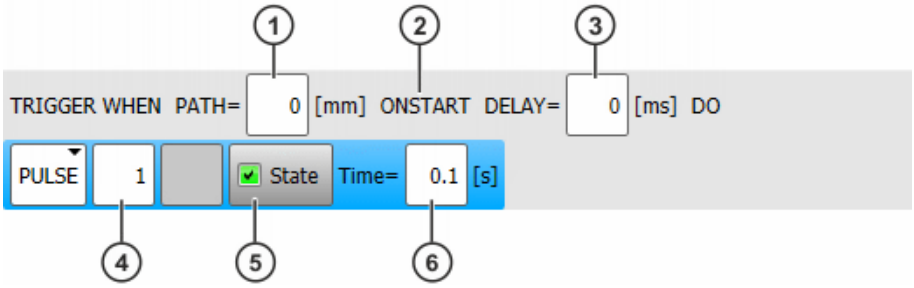


Fig. 10-37: Inline form TRIGGER WHEN PATH, type PULSE

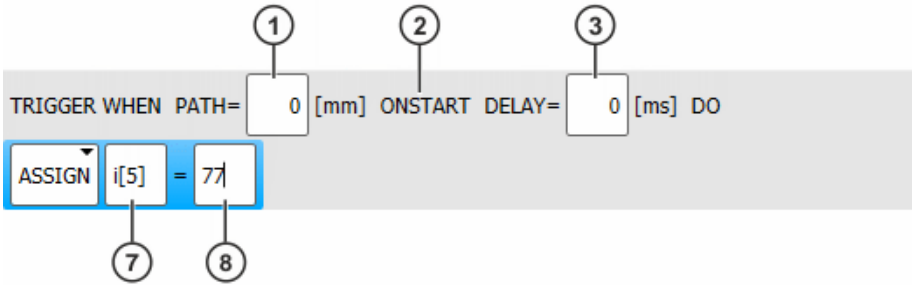


Fig. 10-38: Inline form TRIGGER WHEN PATH, type ASSIGN

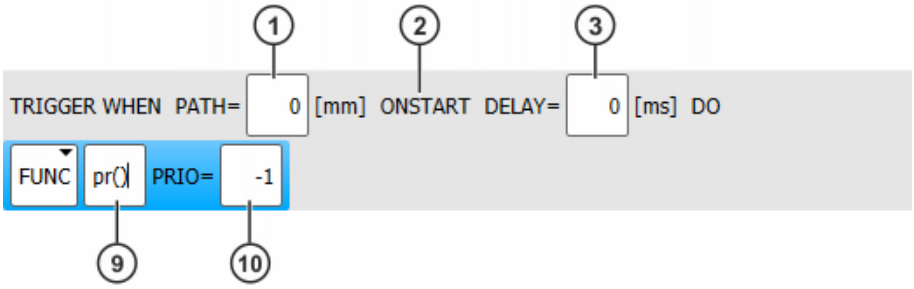


Fig. 10-39: Inline form TRIGGER WHEN PATH, type FUNC

Item	Description
1	Shift in space, starting from the reference point. If no shift in space is desired, enter “0”. • -1,000 … +1,000 mm Negative value: offset towards the start of the motion Positive value: offset towards the end of the motion (>>> *"Max. offset" Page 639*) The shift in space can also be taught instead of being defined here numerically. In this case, the box **ONSTART** is automatically set to **FALSE**. (>>> *10.5.8 "Teaching “Path” for trigger, conditional stop or constant velocity range" Page 501*)
2	Reference point of the trigger • With ONSTART: Start point • Without ONSTART: End point **Toggle OnStart** in the button bar at the bottom can be used to set or cancel ONSTART. (>>> *11.12.2.1 "Reference point for approximate positioning – overview" Page 642*)
3	Shift in time relative to the PATH value. If no shift in time is desired, enter “0”. • -1,000 … +1,000 ms **Toggle Delay** in the button bar at the bottom makes the box editable if this is not already the case. Negative value: offset towards the start of the motion Positive value: trigger is switched after *Time* has elapsed. (>>> *"Max. offset" Page 639*)
4	Number of the digital output • 1 … 8192 If a name has been assigned to the output, it is displayed in the box on the right next to the number.
5	State to which the output is switched • Check box active: “High” level • Check box not active: “Low” level
6	Length of the pulse • 0 … 3.00 s
7	Variable to which a value is to be assigned **Note:** Runtime variables cannot be used.
8	Value to be assigned to the variable
9	Name of the subprogram to be called

Item	Description
10	Priority of the trigger 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.

10.5.6 Programming a conditional stop for spline

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 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}).*

(>>> *10.5.6.3 "Stop condition: example and braking characteristics" Page 496*)

Programming

- At the spline segment or individual spline instruction:

  (>>> *10.5.6.1 "Programming a conditional stop at the spline segment or individual spline instruction" Page 493*)
- For individual spline instruction or for spline block (via a separate inline form):

  (>>> *10.5.6.2 "Inline form STOP WHEN PATH: programming a conditional stop for a an individual spline instruction or spline block" Page 495*)

10.5.6.1 Programming a conditional stop at the spline segment or individual spline instruction

Precondition

- User rights: Function group **New motion range inline forms**
- Program is selected or open.

- T1 mode

Procedure

1. Open the inline form of the spline segment or individual instruction.
2. To display **ADAT** in the inline form (if not already displayed):
   - For segments:
     Select **Switch parameter** > **Spline logic**.
   - For individual instructions:
     Press the **Spline logic** button.
3. Touch the arrow next to **ADAT**: the option window for the logic parameters opens.
4. Select the **Conditional stop** tab.
5. Set the value **ENABLED** in the **Conditional stop** box.
   Further boxes are displayed. Carry out the settings for the conditional stop there.
6. Save the settings with **Cmd OK** and close the inline form.

“Conditional stop” tab

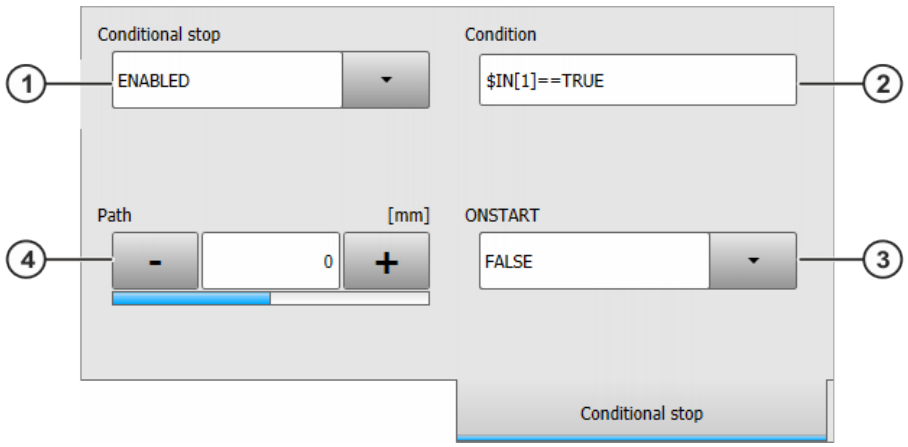


Fig. 10-40: “Conditional stop” tab

Item	Description
1	• **DISABLED** (default): No conditional stop The other boxes are not displayed. • **ENABLED**: Conditional stop
2	Stop condition. The following are permitted: • a global Boolean variable • a signal name • a comparison • a simple logic operation: NOT, OR, AND or EXOR

Item	Description
3	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. (>>> *11.12.2.1 "Reference point for approximate positioning – overview" Page 642*)
4	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 639*) The shift in space can also be taught instead of being defined here numerically. In this case, the box **ONSTART** is automatically set to **FALSE**. (>>> *10.5.8 "Teaching “Path” for trigger, conditional stop or constant velocity range" Page 501*)

10.5.6.2 Inline form STOP WHEN PATH: programming a conditional stop for a an individual spline instruction or spline block

Precondition

- User rights: Function group **New motion range inline forms**
- Program is selected or open.
- T1 mode

There may be other statements between the inline form and the spline instruction, including the motion instructions LIN, CIRC, PTP. The conditional stop always refers to the spline, however.

Procedure

1. Position the cursor in the line after which the conditional stop is to be inserted.
2. Select the menu sequence **Commands** > **Motion parameters** > **Spline Stop Condition**.
3. Set the parameters in the inline form.
4. Press **Cmd OK**.

Inline form STOP WHEN PATH

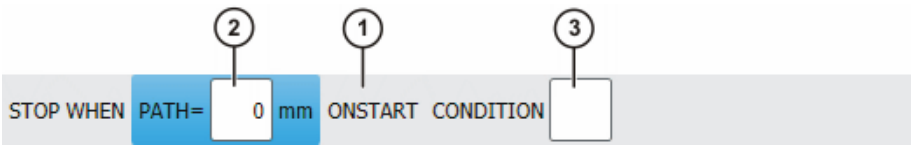


Fig. 10-41: Inline form STOP WHEN PATH

Item	Description
1	Point to which the conditional stop refers • With ONSTART: Start point • Without ONSTART: End point 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 instead of being defined here numerically. In this case, the box **ONSTART** is automatically set to **FALSE**. (>>> *10.5.8 "Teaching “Path” for trigger, conditional stop or constant velocity range" Page 501*)
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

10.5.6.3 Stop condition: example and braking characteristics

Example

The indentations are not present by default and have been inserted here for greater clarity.

The example illustrates programming with inline forms. The principle is the same as that for programming with KRL.

```
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. 10-42: 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.

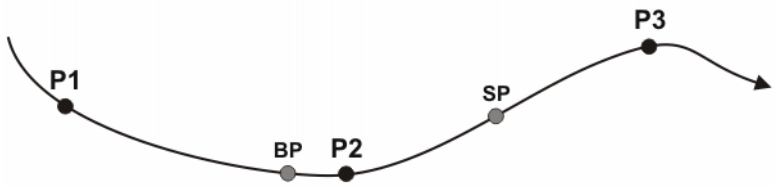


Fig. 10-43: Example of STOP WHEN PATH

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**.

Situation at BP	Behavior of the robot
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.

10.5.7 Programming a constant velocity range for a 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.



A constant velocity range can be programmed in PTP spline blocks.

Precondition

- User rights: Function group **New motion range inline forms**
- Program is selected or open.
- T1 mode
- The spline block fold is open.

Procedure

Program the start of the constant velocity range:

1. Open the inline form of the CP spline segment at which the constant velocity range is to start.
2. To display **ADAT** in the inline form (if not already displayed):
   Select **Switch parameter** > **Spline logic**.
3. Touch the arrow next to **ADAT**: the option window for the logic parameters opens.
4. Select the **Constant velocity** tab.
5. Set the value **START** in the **Constant velocity** box.
   Make further settings if required.
6. Save the settings with **Cmd OK** and close the inline form.

Program the end of the constant velocity range:

1. Open the inline form of the CP spline segment at which the constant velocity range is to end.
2. Perform steps 2 to 6 as for “Program the start of the constant velocity range”, but set **END** in step 5.

“Constant velocity” tab

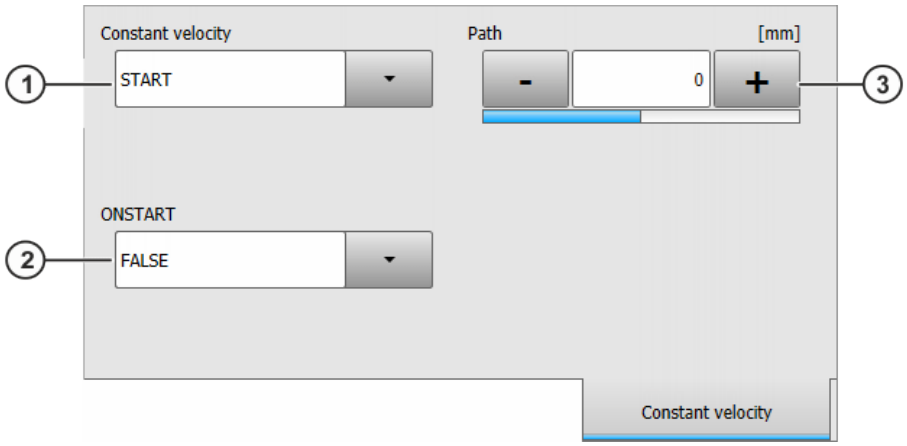


Fig. 10-44: “Constant velocity” tab

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. (>>> *11.12.2.2 "Reference point for homogenous approximate positioning" Page 642*) • **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.

Item	Description
3	The start or end of the constant velocity range can be shifted in space. For this, specify the desired distance. • Positive value: offset towards the end of the motion • Negative value: offset towards the start of the motion The shift in space can also be taught instead of being defined here numerically. In this case, the box **ONSTART** is automatically set to **FALSE**. (>>> *10.5.8 "Teaching “Path” for trigger, conditional stop or constant velocity range" Page 501*)

10.5.7.1 Block selection to a constant velocity range

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.

10.5.7.2 Maximum limits of a constant velocity range

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.

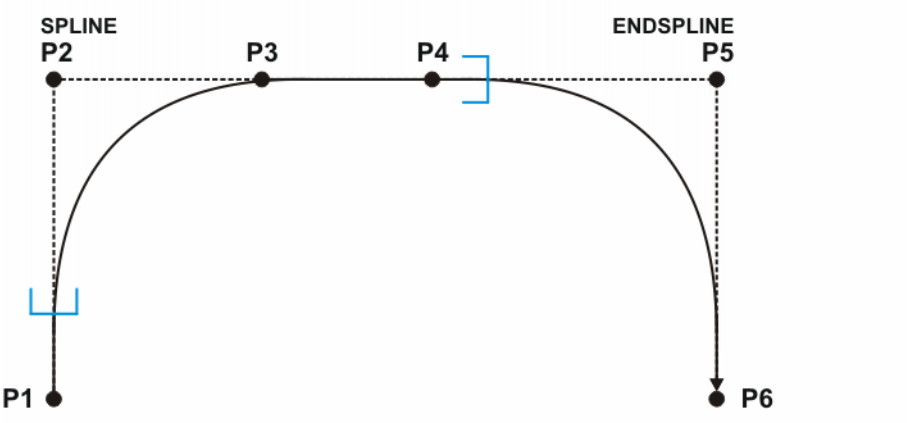


Fig. 10-45: Maximum limits for approximated SPLINE/ENDSPLINE

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.

10.5.8 Teaching “Path” for trigger, conditional stop or constant velocity range

Description

Offsets can be specified for trigger, conditional stop and constant velocity range for the spline. Instead of entering these offsets numerically, they can also be taught.



A taught offset can only ever refer to the end point of the motion. Therefore, if the when the offset is taught, the box **ONSTART** is automatically set to **FALSE**.

Precondition

- User rights: Function group **New motion range inline forms**
- A program is selected.
- T1 mode
- The end point to which the offset refers has already been taught.

Procedure for inline forms TRIGGER WHEN PATH/ STOP WHEN PATH

1. Move the TCP to the desired position.
2. Open the inline form TRIGGER WHEN PATH or STOP WHEN PATH via **Change**.
3. Press **Teach path**.
   The distance from the end point is now applied in the inline form as the value for the offset.
4. Save the change with **Cmd OK**.

Procedure for spline segments or individual spline instructions

1. Move the TCP to the desired position.
2. Open the inline form for the spline segment or individual instruction via **Change**.
3. To display **ADAT** in the inline form (if not already displayed):
   - For segments: Select **Switch parameter** > **Spline logic**.
   - For individual instructions: Press the **Spline logic** button.

The **Spline actions** button, that is required subsequently, is only available if **ADAT** is displayed.

4. Only for triggers:
   a. Touch the arrow next to **ADAT**. The option window for the logic parameters opens.
   b. Select the desired trigger on the **Trigger** tab.
5. Press **Spline actions**. Then select one of the following commands, depending on what the offset is to be taught for:
   - **Record trigger path**
   - **Teach conditional stop path**
   - **Record constant velocity path**

   The distance from the end point is now applied as the value for the offset.
6. Save the change with **Cmd OK**.

10.6 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.

10.7 Modifying programmed motions

10.7.1 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**.

10.7.2 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.

10.7.3 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**.

10.7.4 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.
   (>>> *10.7.4.1 ""Axis mirroring" window" Page 507*)
   (>>> *10.7.4.2 ""Transform - Axis Specific" window" Page 508*)
   (>>> *10.7.4.3 ""Transform - Cartesian Base" window" Page 509*)
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.

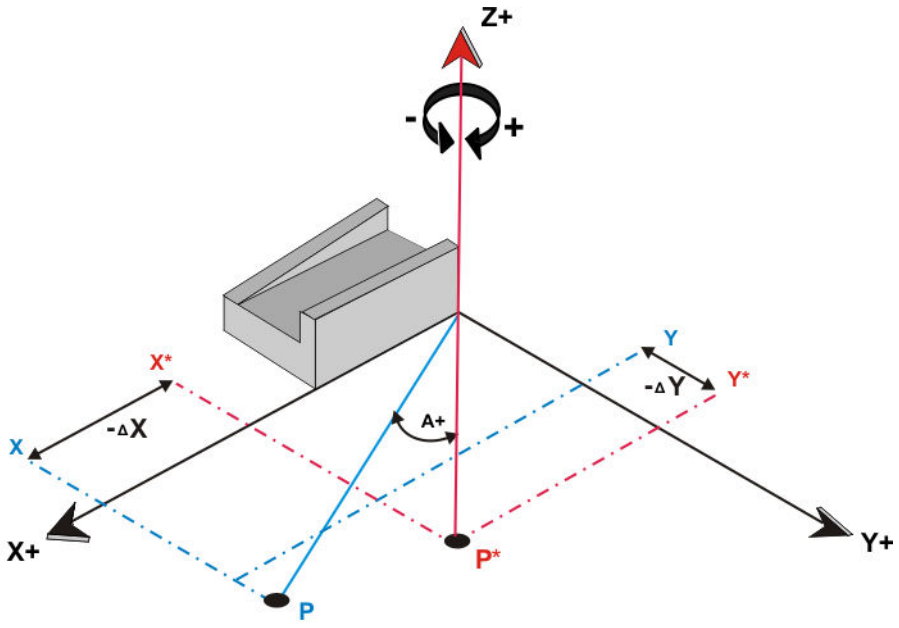


Fig. 10-46: Transform - Cartesian Base

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.

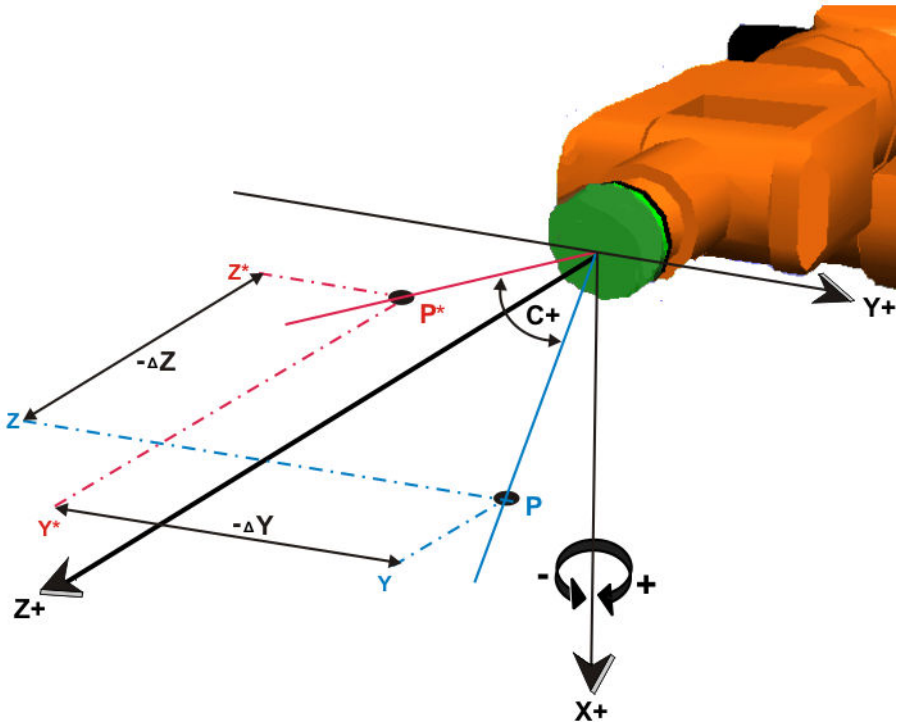


Fig. 10-47: Transform - Cartesian TCP

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.

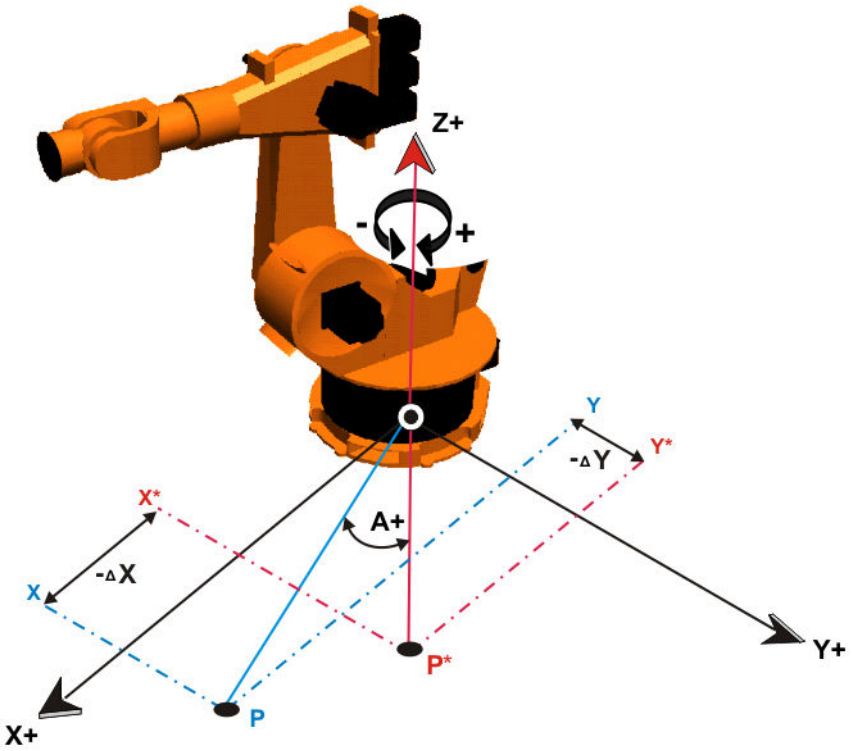


Fig. 10-48: Transform - Cartesian World

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.

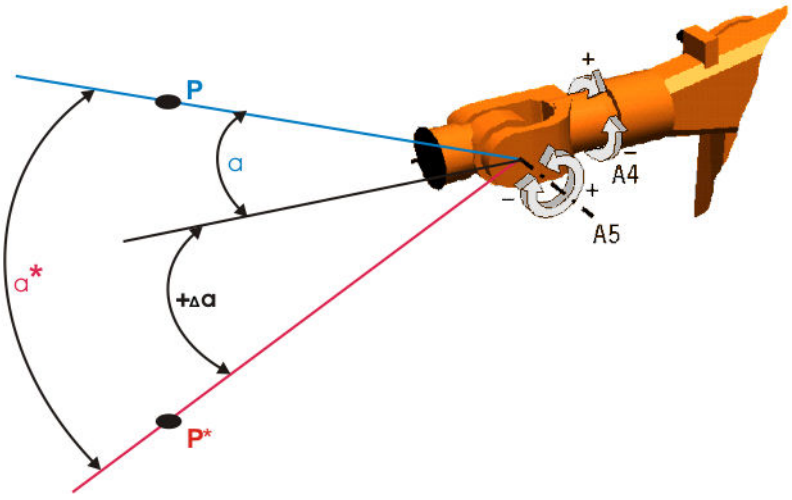


Fig. 10-49: Transform - Axis Specific

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.

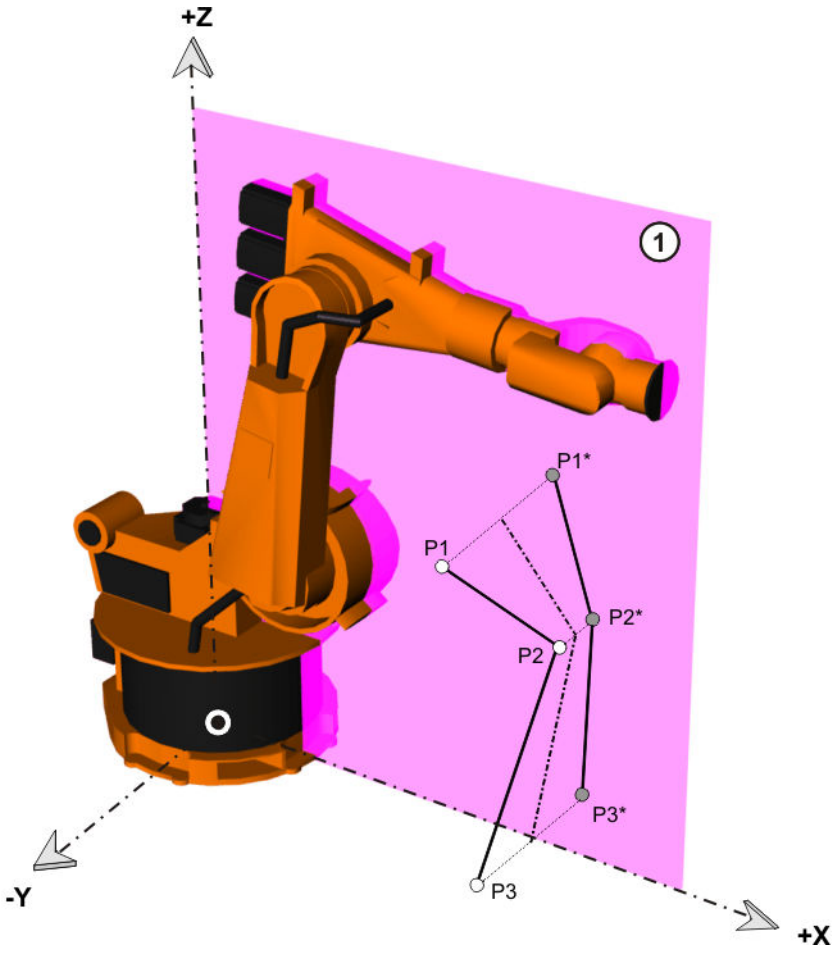


Fig. 10-50: Mirroring

Points P1, P2 and P3 are mirrored in the XY plane (1). The new positions of the points are P1*, P2* and P3*.

10.7.4.1 “Axis mirroring” window

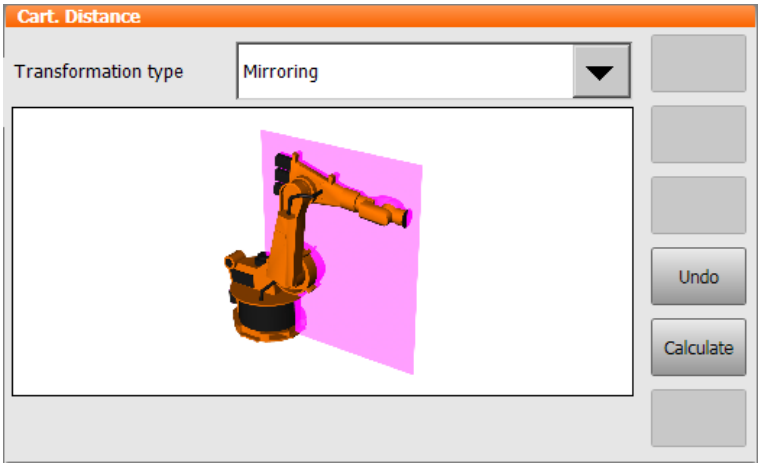


Fig. 10-51: Mirroring

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.

10.7.4.2 "Transform - Axis Specific" window

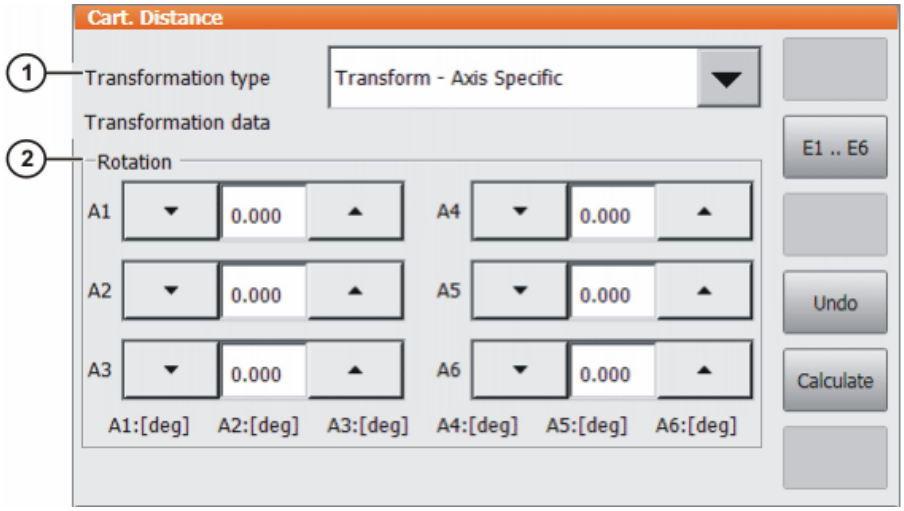


Fig. 10-52: Point transformation - axis-specific

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.

Button	Description
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.

10.7.4.3 "Transform - Cartesian Base" window

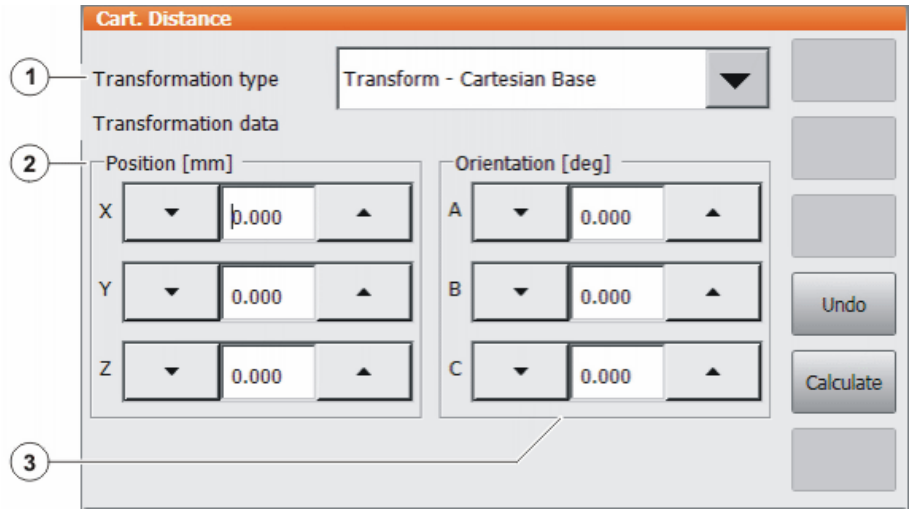


Fig. 10-53: Point transformation - Cartesian

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.

10.8 Global points

10.8.1 Global points – introduction

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. 10-54: 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. 10-55: 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/modify global points**.

10.8.2 Creating a global point (creating new point or converting local point)

Precondition

- User rights: Function group **Teach/modify global points**

> The function group **Teach/modify global points** is deactivated by default. This means that no user group has the right to teach or modify 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 512*)

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. 10-56: Calling point data (example: SLIN)

3. An option window is opened.
   In **Parameter**, activate the check box next to **Global point**.

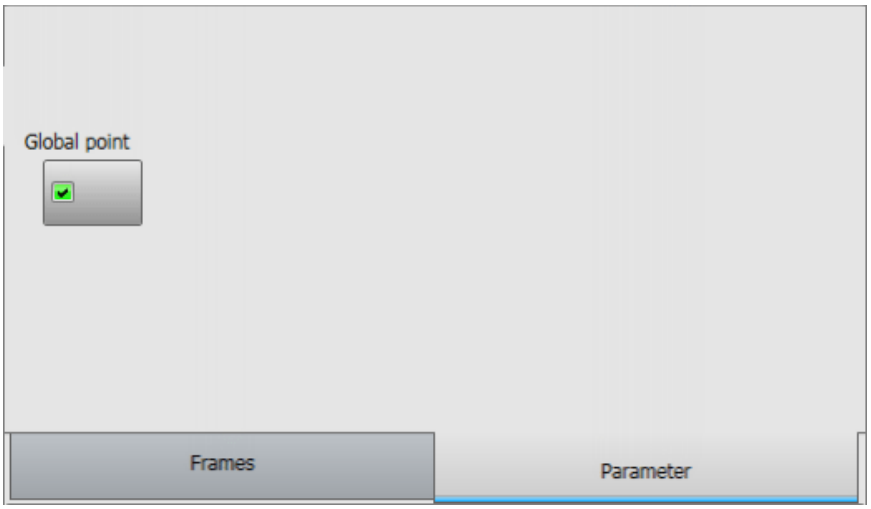


Fig. 10-57: Option window “Parameter”

4. Close the option menu. The point name is now labeled as GLOBAL in the inline form.

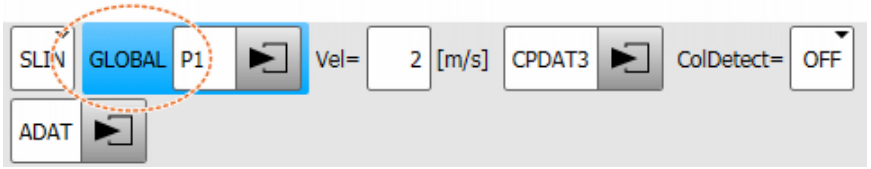


Fig. 10-58: Inline form with global point (using SLIN as an example)

5. Save with **Cmd OK**.

10.8.3 Modifying global points individually

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

Risk of injury due to modification of global points
If a global point is modified, this affects all modules in which it is used. This may result in unexpected changes to the program sequence. These could, in turn, result in injuries or material damage.

- 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, even if the module itself has not been changed.

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.

Modification of tool/base/external TCP:

The modification of tool, base and the setting **External TCP** is only possible in WorkVisual for global points.

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/modify global points**.

The modification affects all modules in which the global point is used. A message with a request for confirmation must be confirmed.

10.8.4 Displaying and modifying global points in the overview

WARNING

Risk of injury due to modification of global points
If a global point is modified, this affects all modules in which it is used. This may result in unexpected changes to the program sequence. These could, in turn, result in injuries or material damage.

- 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, even if the module itself has not been changed.

Precondition

- Only if points are to be modified:
  User rights: Function group **Teach/modify global points**

  The function group **Teach/modify global points** is deactivated by default. This means that no user group has the right to teach or modify global points. A user group must therefore first be assigned to the function group if this has not already been done.

  No special user rights are required for displaying and searching.
- No program is selected.
- T1, T2 or AUT mode

Procedure

1. In the main menu, select **Configuration** > **Global points**.
   The **Global points** window opens.
2. Search for a global point and modify as required.
3. Press **Save**. A request for confirmation is displayed.
4. If the request for confirmation is answered with **Yes**, the changes will be saved.

“Global points” window

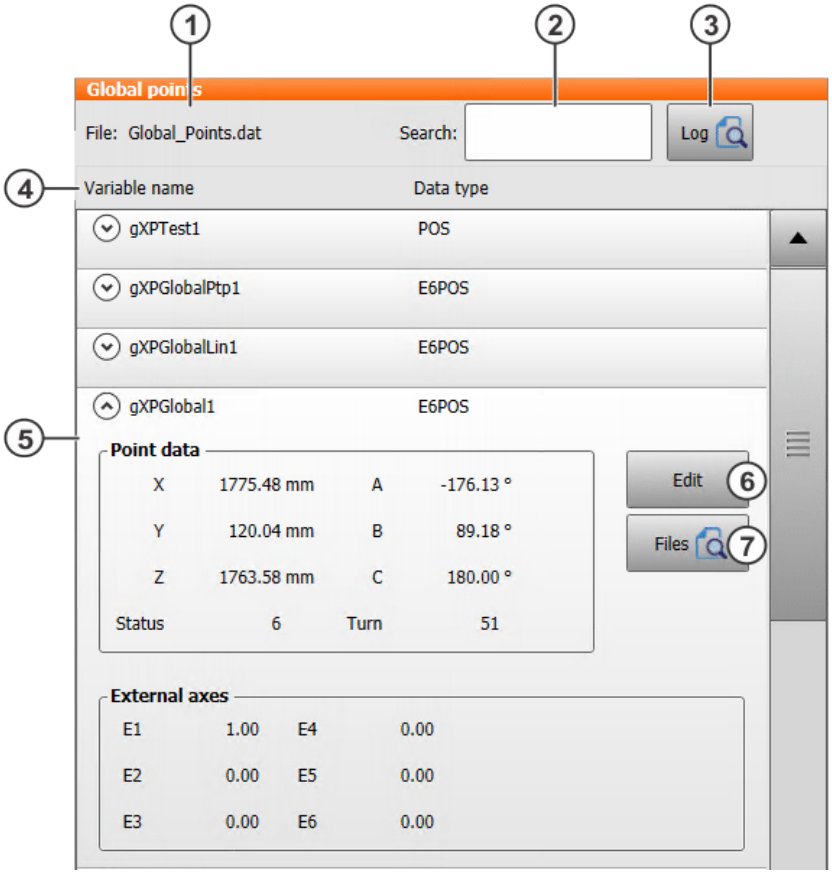


Fig. 10-59: “Global points” window

Item	Description
1	File whose global points are displayed As standard, only Global_Points.dat is available here. Further files can be added. (>>> *10.8.5 "Adding files in the Global points overview" Page 515*)
2	Enter the name or name components for which the display is to be filtered.
3	Archives all global points under C:\KRC\Roboter\LOG\GlobalPointsLogFile.txt. (Irrespective of which file might have been selected under item 1.)
4	List of the points with specification of the variable name and data type List entries can be expanded using the arrow symbol.
5	Expanded entry displays the point data.
6	Opens the list entry in edit mode.
7	Displays a list of all files in which the specific global point is used.

10.8.5 Adding files in the Global points overview

Description

The global points contained in the file Global_Points.dat are displayed as standard in the **Global points** window. The System Software saves global points to Global_Points.dat under KRC:\R1\System by default.

It is also possible to display global points from other DAT files. This is useful, for example, if an option package saves its global points to a file other than Global_Points.dat.

Precondition

- User group “Expert”
- Operating mode T1 or T2.
- No program is selected.

Procedure

Option 1: Expand existing config file

- Enter and save the additional DAT file(s) in GlobalPointFiles.config under C:\KRC\USER.

  As standard, only Global_Points.dat is listed there. The line can be copied and adapted.

```
<?xml version="1.0" encoding="utf-8"?>
<GlobalPointFilesConfiguration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <GlobalPointFileList>
    <GlobalPointFile FileName="Global_Points.dat" />
    <GlobalPointFile FileName="TechGlobals.dat" />
  </GlobalPointFileList>
</GlobalPointFilesConfiguration>
```

Fig. 10-60: Example: TechGlobals.dat has been added

Option 2: Create additional config file

1. Under C:\KRC\USER, copy GlobalPointFiles.config and expand the name, e.g. to GlobalPointsFiles**_Tech**.config.
   The name must start with “GlobalPointsFiles”.

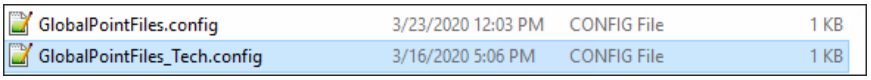


Fig. 10-61: Example: GlobalPointsFiles_Tech.config has been created

2. In the new file, replace the entry “Global_Points.dat” with the desired DAT file, e.g. TechGlobals.dat.

The following applies for both options:

The next time the **Global points** window is opened, the additional DAT file will be available for selection there, in the example TechGlobals.dat.

The precondition for this is that the DAT file is actually present on the robot controller, otherwise it is not available for selection.

10.9 Programming logic instructions

10.9.1 Inline form WAIT: Programming a wait time

Description

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.

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.
4. Save instruction with **Cmd OK**.

Inline form

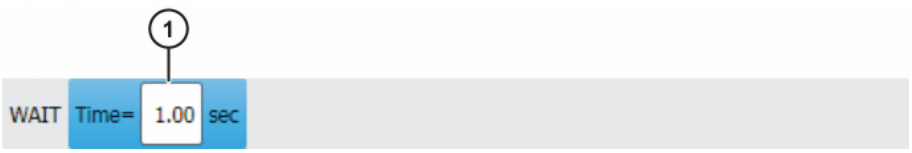


Fig. 10-62: Inline form “WAIT”

Item	Description
1	Wait time • **≥ 0** s

10.9.2 WAIT FOR inline form: programming a signal-dependent wait function

Description

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.

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.
4. Press **Cmd OK** to save the instruction.

Inline form

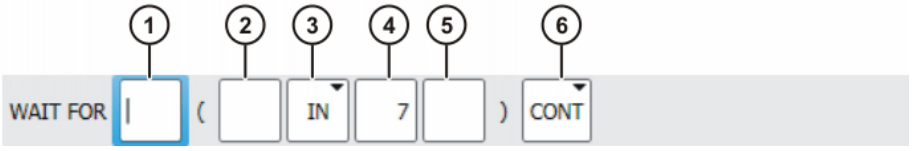


Fig. 10-63: WAIT FOR inline form

Item	Description
1	• **NOT**: add NOT. • **[blank]**
	Add external logic operation. The term is located between the bracketed expressions. Add the desired term by means of the corresponding button: • **AND**, **OR** or **EXOR**
2	• **NOT**: add NOT. • **[blank]**
	Add internal logic operation. The operator is located inside a bracketed expression. Add the desired operator by means of the corresponding button: • **AND**, **OR** or **EXOR**

Item	Description
3	Signal for which the system is waiting. Default selection: • **IN**, **OUT**, **CYCFLAG**, **TIMER** or **FLAG** The **User list**button can be used to display a list of user-defined variables instead of the default selection. Precondition: the list has been configured. (>>> *6.20 "Configuring “user list” with variables for inline forms" Page 273*) **System list** displays the default selection again. It is possible to toggle between the buttons by touching them.
4	• If a signal from the default selection has been selected: enter number. • If a variable from the user list has been selected: enter value.
5	This box is only available if a signal from the default selection has been selected. If a name exists for the signal, this name is displayed. User group “Expert” or higher: a name can be entered or the existing name can be changed. As long as the inline form has not been saved, these changes can be reset again by pressing **Long text**.
6	• **CONT**: execution in the advance run • **[blank]**: execution with advance run stop

10.9.3 Inline form OUT: Setting a digital output

Description

The instruction sets a digital output.

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.
4. Save instruction with **Cmd OK**.

Inline form

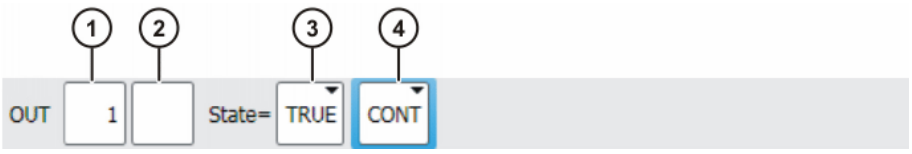


Fig. 10-64: Inline form “OUT”

Item	Description
1	Number of the digital output • **1 … 8192**
2	If a name exists for the output, this name is displayed. User group “Expert” or higher: A name can be entered or the existing name can be changed. As long as the inline form has not been saved, these changes can be reset again by pressing **Long text**.
3	State to which the output is switched • **TRUE** • **FALSE**
4	• **CONT**: Execution in the advance run • **[blank]**: Execution with advance run stop

10.9.4 Inline form PULSE: Setting a pulse output

Description

The instruction sets a pulse of a defined length.

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.
4. Save instruction with **Cmd OK**.

Inline form

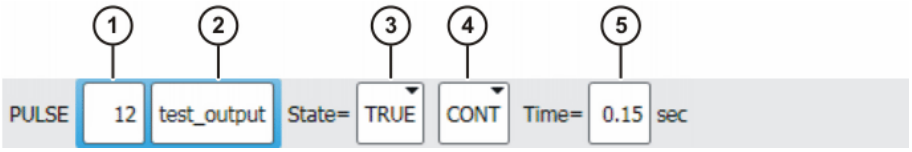


Fig. 10-65: Inline form “PULSE”

Item	Description
1	Number of the digital output • **1 … 8192**
2	If a name exists for the output, this name is displayed. User group “Expert” or higher: A name can be entered or the existing name can be changed. As long as the inline form has not been saved, these changes can be reset again by pressing **Long text**.

Item	Description
3	State to which the output is switched • **TRUE**: “High” level • **FALSE**: “Low” level
4	• **CONT**: Execution in the advance run • **[blank]**: Execution with advance run stop
5	Length of the pulse • **0.10 … 3.00 s**

10.9.5 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.
4. Save instruction with **Cmd OK**.

10.9.5.1 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.

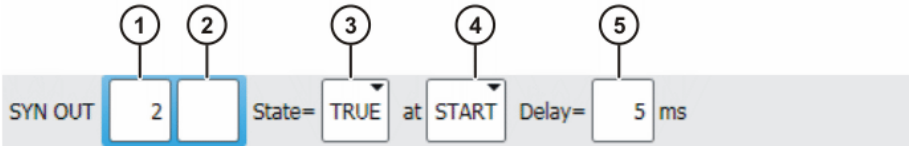


Fig. 10-66: Inline form “SYN OUT”, option “START/END”

Item	Description
1	Number of the digital output • **1 … 8192**
2	If a name exists for the output, this name is displayed. User group “Expert” or higher: A name can be entered or the existing name can be changed. As long as the inline form has not been saved, these changes can be reset again by pressing **Long text**.

Item	Description
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
```

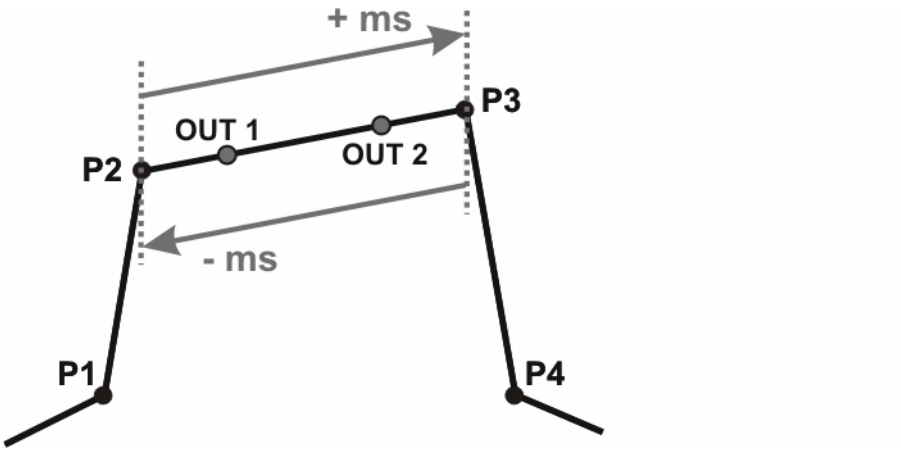


Fig. 10-67

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
```

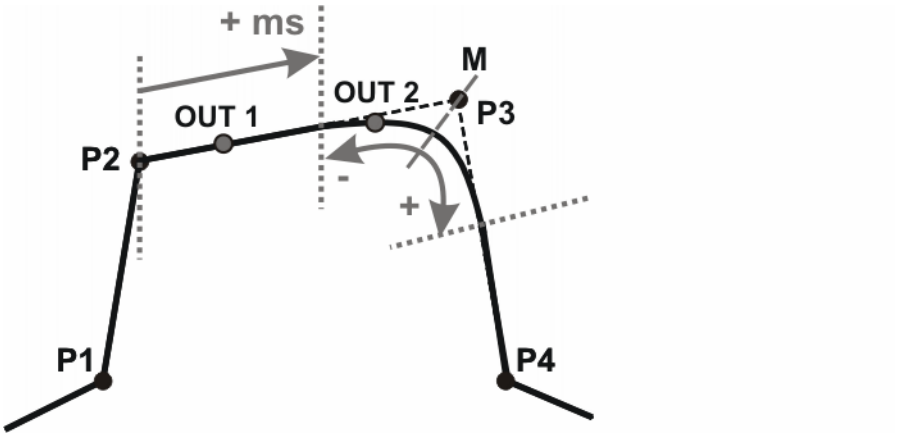


Fig. 10-68

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
```

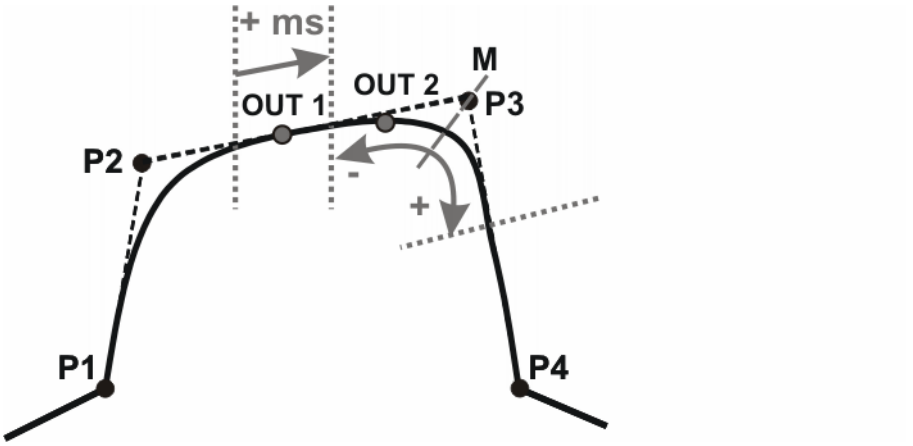


Fig. 10-69

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.

10.9.5.2 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.

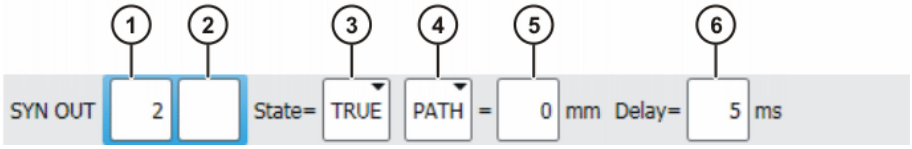


Fig. 10-70: Inline form “SYN OUT”, option “PATH”

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 or the existing name can be changed. As long as the inline form has not been saved, these changes can be reset again by pressing **Long text**.
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
```

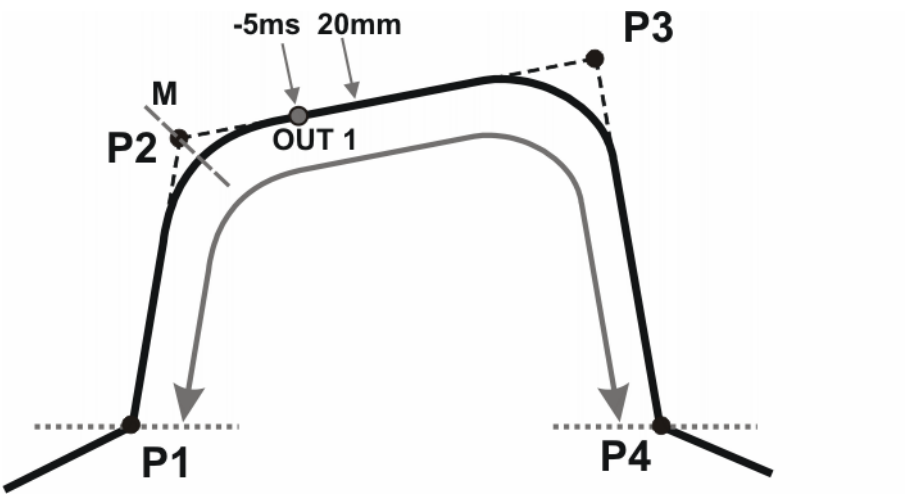


Fig. 10-71

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
```

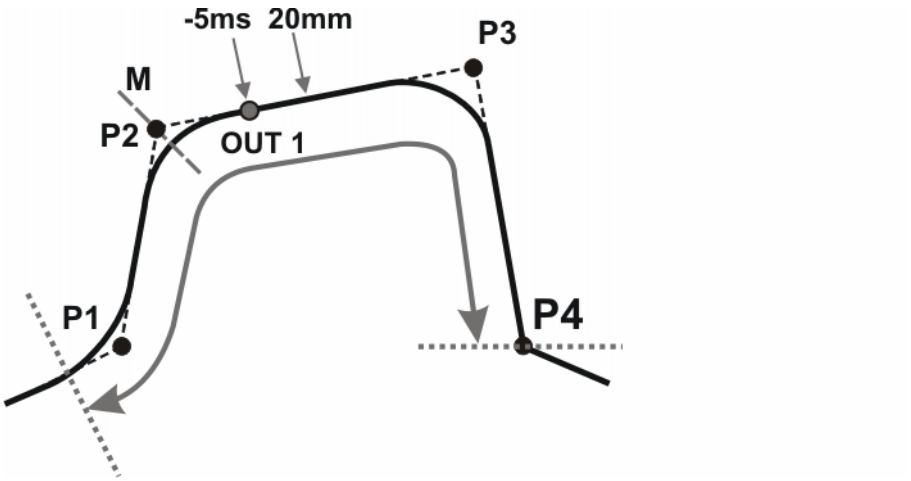


Fig. 10-72

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.

10.9.6 Inline form SYN PULSE: Setting a pulse on the path

Description

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.

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.
4. Save instruction with **Cmd OK**.

Inline form

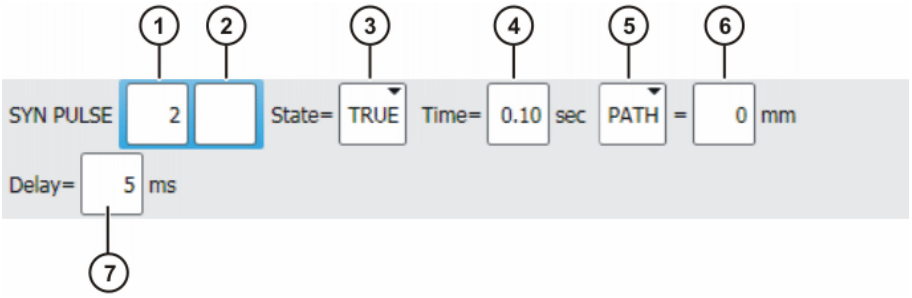


Fig. 10-73: Inline form “SYN PULSE”

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 or the existing name can be changed. As long as the inline form has not been saved, these changes can be reset again by pressing **Long text**.
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. • **PATH**: SYN PULSE refers to the end point. An offset in space is also possible. See SYN OUT for examples and switching limits.
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.

10.9.7 Setting an analog output

Description

The robot controller can manage 32 analog inputs and 32 analog outputs. The inputs/outputs are managed via the following system variables:

	Inputs	Outputs
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}*

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.
   (>>> *10.9.7.1 "Inline form “ANOUT” (static)" Page 527*)
   (>>> *10.9.7.2 "Inline form “ANOUT” (dynamic)" Page 527*)
4. Save instruction with **Cmd Ok**.

10.9.7.1 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.

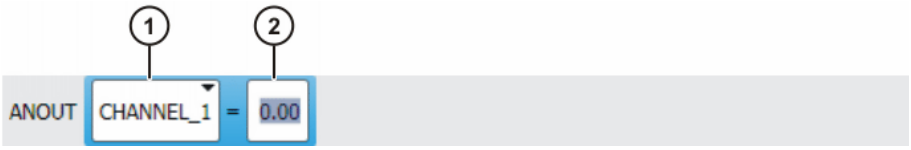


Fig. 10-74: Inline form “ANOUT” (static)

Item	Description
1	Number of the analog output • **CHANNEL_1 … CHANNEL_32**
2	Factor for the voltage • **0 … 1** (intervals: 0.01)

10.9.7.2 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.

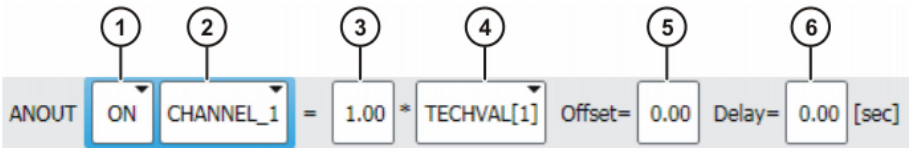


Fig. 10-75: Inline form “ANOUT” (dynamic)

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**

10.9.8 Inline form TIMER: Programming a timer

Precondition

- Program is selected or open.
- Operating mode T1

Procedure

1. Position the cursor in the line after which the instruction is to be inserted.
2. Select the menu sequence **Commands** > **Logic** > **TIMER**.
3. Set the parameters in the inline form.
4. Press **Cmd OK** to save the instruction.

Inline form



Fig. 10-76: Inline form “TIMER”

Item	Description
1	Number of the timer • **1 … 64**
2	This box is only present if Action = TIMOUT: Starting value for the timeout [ms] • **-100 000 … -1**
3	• **ON**: Starts the timer. • **OFF**: Stops the timer. • **TIMEOUT**: Starts the timer with a negative value. When the timer crosses zero, $TIMER_FLAG[*no.*] changes from FALSE to TRUE. (>>> *10.9.8.1 "$TIMER_FLAG" Page 529*)

10.9.8.1 $TIMER_FLAG

Description

Flag for the timer

The variable indicates whether the value of the timer is greater than or equal to zero.

$TIMER_FLAG can be used in wait or interrupt conditions that are to be triggered after a certain time has elapsed. If the corresponding timer is started with a negative value, $TIMER_FLAG changes from FALSE to TRUE at the zero crossing.

Syntax

`$TIMER_FLAG[`*Index*`]=`*State*

Explanation of the syntax

Element	Description
Index	Type: INT Number of the timer • **1 … 64**
Status	Type: BOOL • TRUE: value greater than or equal to zero • FALSE: value less than zero

Example

```
$TIMER[4] = -50
$TIMER_STOP[4] = FALSE
```

```
WAIT FOR $IN[1] or $TIMER_FLAG[4]
```

10.9.9 Inline form “Variable declarations”

Description

Inline form for declaring a variable. If necessary, the variable can be initialized straight away.

Precondition

- Program has been opened.
- T1 mode

Procedure

1. Position the cursor in any line.
2. Select the menu sequence **Commands** > **Logic** > **Variable declarations**.
3. If the variable is to be initialized immediately, display the box **Varvalue** by pressing the **Initialize** button.
4. Set the parameters in the inline form.
5. Press **Cmd OK** to save the instruction.
   - The declaration is inserted into the program beneath the DEF line.
   - When **Varvalue** is displayed, the variable is initialized in the INI fold, USER INI sub-fold.

Inline form

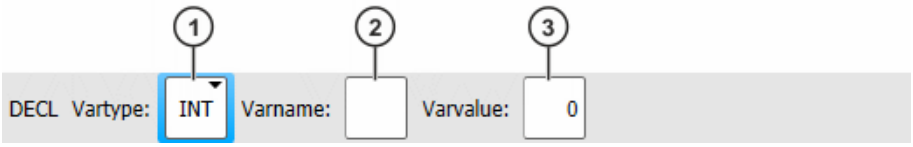


Fig. 10-77: Inline form “Variable declarations” with Varvalue box

Item	Description
1	Specify data type: • **INT**, **BOOL**, **REAL** or **CHAR**
2	Specify the name: max. 24 characters
3	If required, this box can also be displayed using the **Initialize** button. Enter the initialization value.

Restrictions for “Change”

It is not possible to modify or add an initialization using the **Change** button.

Possible modifications: Changing the name and type of the variable

10.9.10 Inline form “Interrupt declaration”

Description

Inline form for declaring an interrupt. The interrupt program can be created automatically at once.

A maximum of 64 interrupts may be declared simultaneously. An interrupt declaration may be overwritten by another at any time.

Interrupt:

In the case of a defined event, e.g. an input, the controller interrupts the current program and executes a defined subprogram.

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.

No interruption of current motions:

Current and prepared robot motions are not generally interrupted by an interrupt. Parallel to the interrupt program, the robot continues to execute all motions that have already been prepared in the main program.

If the interrupt program is completely executed during this time, the main program is resumed without any pause between motions, i.e. without the processing time being lengthened.



Interrupt programs must not contain any spline motions.

Precondition

- Program has been opened.
- T1 mode

Procedure

1. Position the cursor in the line after which the code is to be inserted.
2. Select the menu sequence **Commands** > **Logic** > **Interrupt declaration**.
3. Set the parameters in the inline form.
4. Press **Cmd OK** to save the instruction.
5. Answer the request for confirmation *The data list will be modified irrevocably. …* with **Yes**.
6. If the interrupt program specified in the inline form does not yet exist, the following query is now displayed:

   The interrupt program is not defined in this module (global functions are not checked). Do you want to create it automatically?

   - If the answer is **Yes**, the program is created automatically.
   - If the answer is **No**, the program must be created subsequently.



Following declaration, an interrupt is initially inactive. The interrupt must be activated via the inline form **Switch the interrupt** before it can respond to the defined event.
(>>> *10.9.11 "Inline form “Switch the interrupt”" Page 533*)

Inline form

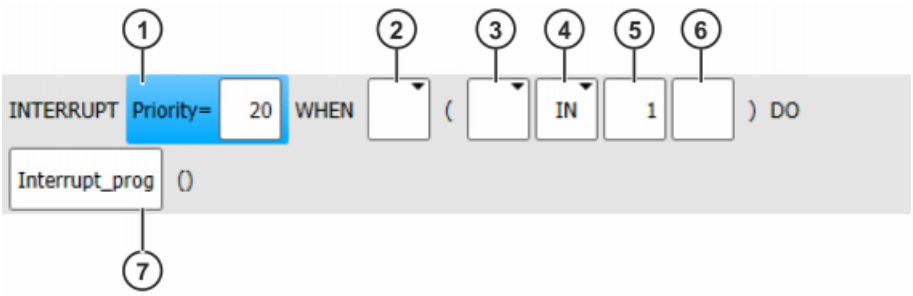


Fig. 10-78: Inline form: Interrupt declaration with “System list”

Item	Description
1	Number (= priority) of the interrupt 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. • **1, 2, 4 … 39 and 81 … 128**
2	• **NOT**: add NOT. • **[blank]**
	Add external logic operation. The term is located between the bracketed expressions. Add the desired term by means of the corresponding button: • **AND**, **OR** or **EXOR**
3	• **NOT**: add NOT. • **[blank]**
	Add internal logic operation. The operator is located inside a bracketed expression. Add the desired operator by means of the corresponding button: • **AND**, **OR** or **EXOR**
4	Event/signal that is evaluated. An empty box in which KRL can be programmed is displayed as standard. Predefined signals can also be selected. The following buttons are available for this purpose. Touching the button will display the next one in each case. • **User list**: Displays a list with user-defined variables. Precondition: The list has been configured. (>>> *6.20 "Configuring “user list” with variables for inline forms" Page 273*) • **System list**: Displays a default selection: **IN**, **OUT**, **CYCFLAG**, **TIMER** or **FLAG** • **Free text**: Displays the box again in which KRL can be entered.
5	• If a signal from the default selection has been selected: enter number. • If a variable from the user list has been selected: enter value.

Item	Description
6	This box is only available if a signal from the default selection has been selected. If a name exists for the signal, this name is displayed. User group “Expert” or higher: a name can be entered or the existing name can be changed. As long as the inline form has not been saved, these changes can be reset again by pressing **Long text**.
7	The name of the interrupt program to be executed: max. 24 characters When the inline form is closed, the controller checks whether the specified interrupt program exists in the opened module. If this is not the case, a dialog is displayed, which can be used to have the program created automatically.

10.9.11 Inline form “Switch the 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.

	Up to 32 interrupts may be active at any one time.

Precondition

- Program is selected or open.
- Operating mode T1

Procedure

1. Position the cursor in the line after which the code is to be inserted.
2. Select the menu sequence **Commands** > **Logic** > **Switch the interrupt**.
3. Set the parameters in the inline form.
4. Press **Cmd OK** to save the instruction.

Inline form

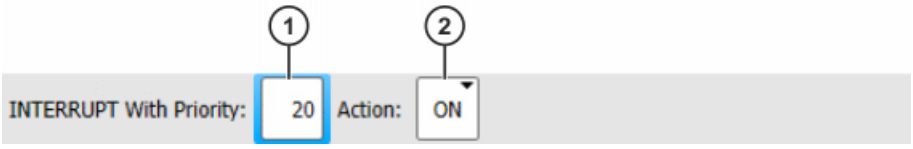


Fig. 10-79: Inline form “Switch the interrupt”

Item	Description
1	Number (= priority) of the interrupt • **1, 2, 4 … 39 and 81 … 128**
2	• **ON**: activates the interrupt. • **OFF**: deactivates the interrupt.

10.9.12 Inline form IF ... THEN: 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.

Precondition

- Program has been opened.
- T1 mode

Procedure

1. Position the cursor in the line after which the code is to be inserted.
2. Select the menu sequence **Commands** > **Logic** > **IF ... THEN**.
3. Set the parameters in the inline form.
4. Press **Cmd OK** to save the instruction.
   The code for the branch is inserted into the program.
5. Now program the desired instructions in the THEN block and, if necessary, in the ELSE block.

Inline form

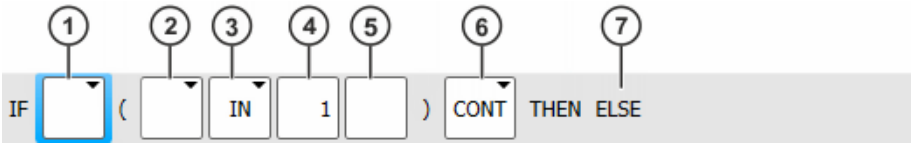


Fig. 10-80: Inline form “IF ... THEN”

Item	Description
1	• **NOT**: Add NOT. • **[blank]**
	Add external logic operation. The term is located between the bracketed expressions. Add the desired term by means of the corresponding button: • **AND**, **OR** or **EXOR**
2	• **NOT**: Add NOT. • **[blank]**
	Add internal logic operation. The operator is located inside a bracketed expression. Add the desired operator by means of the corresponding button: • **AND**, **OR** or **EXOR**

Item	Description
3	Signal that is evaluated. Default selection: • **IN**, **OUT**, **CYCFLAG**, **TIMER** or **FLAG** Other signals can also be programmed. The following buttons are available for this purpose. Touching the button will display the next one in each case. • **Free text**: Displays a box in which KRL can be entered. • **User list**: Displays a list with user-defined variables. Precondition: The list has been configured. (>>> *6.20 "Configuring “user list” with variables for inline forms" Page 273*) • **System list**: Displays the default selection again.
4	• If a signal from the default selection has been selected: enter number. • If a variable from the user list has been selected: enter value.
5	This box is only available if a signal from the default selection has been selected. If a name exists for the signal, this name is displayed. User group “Expert” or higher: A name can be entered or the existing name can be changed. As long as the inline form has not been saved, these changes can be reset again by pressing **Long text**.
6	• **CONT**: Execution in the advance run • **[blank]**: Execution with advance run stop
7	Add **ELSE** by means of the corresponding button. The focus must be on box 1 or 2.

Restrictions for “Change”

Only the condition can be modified by means of the **Change** button. ELSE or CONT cannot be added or removed.

10.9.13 Inline form 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.

Precondition

- Program has been opened.
- T1 mode

Procedure

1. Position the cursor in the line after which the code is to be inserted.

2. Select the menu sequence **Commands** > **Logic** > **WHILE ... END-WHILE**.
3. Set the parameters in the inline form.
4. Press **Cmd OK** to save the instruction.
   The code for the loop is inserted into the program.
5. Now program the desired instructions between WHILE and END-WHILE.

Inline form

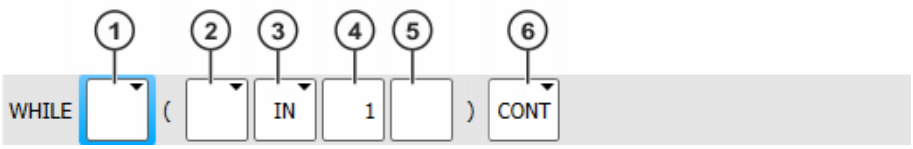


Fig. 10-81: Inline form “WHILE ... ENDWHILE”

Item	Description
1	• **NOT**: Add NOT. • **[blank]**
	Add external logic operation. The term is located between the bracketed expressions. Add the desired term by means of the corresponding button: • **AND**, **OR** or **EXOR**
2	• **NOT**: Add NOT. • **[blank]**
	Add internal logic operation. The operator is located inside a bracketed expression. Add the desired operator by means of the corresponding button: • **AND**, **OR** or **EXOR**
3	Signal that is evaluated. Default selection: • **IN**, **OUT**, **CYCFLAG**, **TIMER** or **FLAG** Other signals can also be programmed. The following buttons are available for this purpose. Touching the button will display the next one in each case. • **Free text**: Displays a box in which KRL can be entered. • **User list**: Displays a list with user-defined variables. Precondition: The list has been configured. (>>> *6.20 "Configuring “user list” with variables for inline forms" Page 273*) • **System list**: Displays the default selection again.
4	• If a signal from the default selection has been selected: enter number. • If a variable from the user list has been selected: enter value.

Item	Description
5	This box is only available if a signal from the default selection has been selected. If a name exists for the signal, this name is displayed. User group “Expert” or higher: A name can be entered or the existing name can be changed. As long as the inline form has not been saved, these changes can be reset again by pressing **Long text**.
6	• **CONT**: Execution in the advance run • **[blank]**: Execution with advance run stop

Restrictions for “Change”

Only the condition can be modified by means of the **Change** button. CONT cannot be added or removed.

10.9.14 Inline form 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.

Precondition

- Program has been opened.
- T1 mode

Procedure

1. Position the cursor in the line after which the code is to be inserted.
2. Select the menu sequence **Commands** > **Logic** > **REPEAT ... UNTIL**.
3. Set the parameters in the inline form.
4. Press **Cmd OK** to save the instruction.
   The code for the loop is inserted into the program.
5. Now program the desired instructions between REPEAT and UNTIL.

Inline form

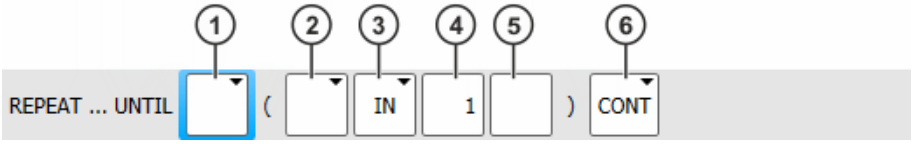


Fig. 10-82: Inline form “REPEAT ... UNTIL”

Item	Description
1	• **NOT**: Add NOT. • **[blank]**
	Add external logic operation. The term is located between the bracketed expressions. Add the desired term by means of the corresponding button: • **AND**, **OR** or **EXOR**
2	• **NOT**: Add NOT. • **[blank]**
	Add internal logic operation. The operator is located inside a bracketed expression. Add the desired operator by means of the corresponding button: • **AND**, **OR** or **EXOR**
3	Signal that is evaluated. Default selection: • **IN**, **OUT**, **CYCFLAG**, **TIMER** or **FLAG** Other signals can also be programmed. The following buttons are available for this purpose. Touching the button will display the next one in each case. • **Free text**: Displays a box in which KRL can be entered. • **User list**: Displays a list with user-defined variables. Precondition: The list has been configured. (>>> *6.20 "Configuring “user list” with variables for inline forms" Page 273*) • **System list**: Displays the default selection again.
4	• If a signal from the default selection has been selected: enter number. • If a variable from the user list has been selected: enter value.
5	This box is only available if a signal from the default selection has been selected. If a name exists for the signal, this name is displayed. User group “Expert” or higher: A name can be entered or the existing name can be changed. As long as the inline form has not been saved, these changes can be reset again by pressing **Long text**.
6	• **CONT**: Execution in the advance run • **[blank]**: Execution with advance run stop

Restrictions for “Change”

Only the condition can be modified by means of the **Change** button. CONT cannot be added or removed.

10.9.15 Inline form 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.

Precondition

- Program has been opened.
- T1 mode

Procedure

1. Position the cursor in the line after which the code is to be inserted.
2. Select the menu sequence **Commands** > **Logic** > **LOOP ... END-LOOP**.
3. If necessary, display the boxes by pressing the **Add abort cond.** button, then set the parameters in the inline form.
4. Press **Cmd OK** to save the instruction.
5. Now program the desired instructions between LOOP and ENDLOOP.

   If no parameters have been set in step 3, at least the EXIT instruction must be programmed, otherwise the loop will be executed endlessly.

Inline form



Fig. 10-83: Inline form “LOOP ... ENDLOOP” without boxes

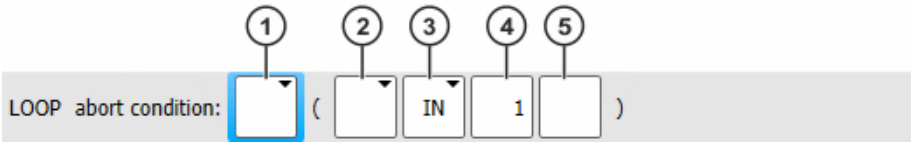


Fig. 10-84: Inline form “LOOP ... ENDLOOP” with boxes

Press the **Add abort cond.** button in order to display the boxes.

Item	Description
1	• **NOT**: Add NOT. • **[blank]**
	Add external logic operation. The term is located between the bracketed expressions. Add the desired term by means of the corresponding button: • **AND**, **OR** or **EXOR**
2	• **NOT**: Add NOT. • **[blank]**
	Add internal logic operation. The operator is located inside a bracketed expression. Add the desired operator by means of the corresponding button: • **AND**, **OR** or **EXOR**

Item	Description
3	Signal that is evaluated. Default selection: • **IN**, **OUT**, **CYCFLAG**, **TIMER** or **FLAG** Other signals can also be programmed: The **Free text** button can be used to display a box in which KRL can be entered instead of the default selection. **System list** displays the default selection again. It is possible to toggle between the buttons by touching them.
4	This box is only available if a signal from the default selection has been selected. Enter the number of the signal.
5	This box is only available if a signal from the default selection has been selected. If a name exists for the signal, this name is displayed. User group “Expert” or higher: A name can be entered or the existing name can be changed. As long as the inline form has not been saved, these changes can be reset again by pressing **Long text**.

Restrictions for “Change”

The inline form cannot be modified by means of the **Change** button!

10.9.16 Inline form FOR ... 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.

Precondition

- Program has been opened.
- T1 mode

Procedure

1. Position the cursor in the line after which the code is to be inserted.
2. Select the menu sequence **Commands** > **Logic** > **FOR ... ENDFOR**.
3. Set the parameters in the inline form.
4. Press **Cmd OK** to save the instruction.
   - If the counter variable has not yet been declared, the following query is displayed: *Do you want to declare a variable with the name "{Name}"?*

     If the response is **Yes**, the variable is declared automatically. If the response is **No**, it must subsequently be declared manually.

   The code is inserted into the program.
5. Now program the desired instructions between FOR and ENDFOR.

Inline form

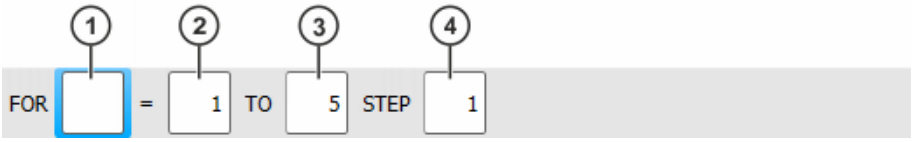


Fig. 10-85: Inline form “FOR ... ENDFOR”

Item	Description
1	INT variable that counts the number of times the loop has been executed - the “counter variable”. The variable can be declared beforehand but does not have to be. The value of the variable can be used in statements inside and outside of the loop. After exiting the loop, the variable retains its most recent value.
2	Start value The counter variable is automatically preset to the start value.
3	End value If the value exceeds or falls below the end value, the loop is terminated.
4	Increment Each time the loop is executed, the counter variable is automatically increased by the increment. The value may be negative. Default: 1. • Positive value: the loop is ended if the counter variable is greater than the end value. • Negative value: the loop is ended if the counter variable is less than the end value. The value may not be either zero or a variable.

10.9.17 Inline form SWITCH ... CASE: Programming a multiple branch

Description

Executes 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 executed.

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.

Precondition

- Program has been opened.
- T1 mode

Procedure

1. Position the cursor in the line after which the code is to be inserted.
2. Select the menu sequence **Commands** > **Logic** > **SWITCH ... CASE**.

3. Set the parameters in the inline form.
4. Press **Cmd OK** to save the instruction.
   - If the selection criterion has not yet been declared as a variable, the following query is displayed: *Do you want to declare a variable with the name "{Name}"?*

     If the response is **INT** or **CHAR**, then the variable is declared automatically.

     In the case of **No**, the variable must subsequently be declared manually (= only option for ENUM variables).

   The code is inserted into the program.
5. Now program the desired statments in the statement blocks.

Inline form

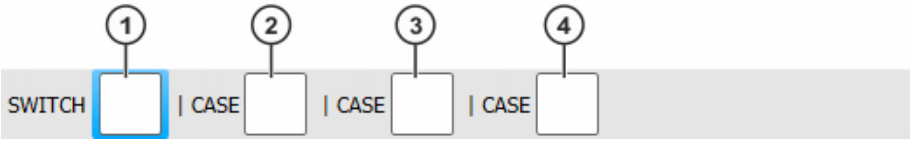


Fig. 10-86: Inline form “SWITCH ... CASE”

Item	Description
1	Variable for the selection criterion • An INT or CHAR variable can be declared beforehand but does not have to be. • An ENUM variable must be declared manually beforehand or afterwards.
2	Identifier for the statement block The data type of the identifier must match the data type of the variable (item 1). A statement block can have any number of identifiers. Multiple identifiers in a CASE array must be separated from each other by a comma.
3	Same as item 2.
4	Existing CASE arrays must either be filled with identifiers or removed. It is possible to add or remove CASE arrays by pressing the corresponding buttons.

Restrictions for “Change”

It is not possible to add or remove a CASE using the **Change** button.

Possible modifications:

- If the cursor is situated in the line with SWITCH, the variable can be modified.
- If the cursor is situated in a line with CASE, the value can be modified.

10.9.18 Modifying a logic instruction



Not all logic instructions can be modified as described here.
- The preconditions specified for the individual instructions apply.
- It is not always possible to modify all parameters. Restrictions are specified in the descriptions of the individual instructions.

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

Damage to the energy supply system due to unfavorable motion sequence

If motions place excessive tension or pressure on the energy supply system, damage to property may result. The energy supply system, for example, can become wound up and then tear off.

- When programming and testing, ensure that the energy supply system is not damaged during the motion sequence.

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 upper-case and lower-case 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. It is therefore recommended that keywords are never used other than with their defined meaning in order to avoid problems (error messages, compiler problems, confusion with existing commands or data types, etc.).

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	ENDFOR
BOOL	ENDIF
CHAR	ENDLOOP
CAST_FROM	ENDSWITCH
CAST_TO	ENDWHILE
CCLOSE	EXT
CHANNEL	EXTFCT
CIOCTL	FALSE
CONFIRM	FF
CONST	FRAME
COPEN	GLOBAL
CREAD	INT
CWRITE	MAXIMUM
DEF	MINIMUM
DEFAULT	POS
DEFDAT	PRIO
DEFFCT	PUBLIC
E6AXIS	SREAD
E6POS	SWRITE
END	REAL
ENDDAT	TRUE
ENDFCT	

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 • **-2³¹-1 … 2³¹-1** Examples: 1; 32; 345
Real	REAL	Floating-point number • **-3.4E+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, INT S, T, REAL E1, E2,
E3, E4, E5, E6
```

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 549*)
- (>>> *11.3.4.2 "Making variables, constants, signals and user data types available globally" Page 550*)

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 CONST: 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`	The keyword GLOBAL must only be used in data lists. (>>> *11.3.4 "Areas of validity" Page 549*)
`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 549*) **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 549*) **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	*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 562*)

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	*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 562*)

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 443*)
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 562*)

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 564*)

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 562*)
`#BASE`, `#TOOL`	Only permissible if the end point was specified in Cartesian coordinates. • #BASE (default): The end point refers to the coordinate system that belongs to the physical base. • #TOOL: The 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 564*)

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	*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 562*)
`#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.

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 564*)

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 443*)

Element	Description
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 562*)
#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

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.

Information about which parameter can be used in which motion instruction can be found in the descriptions of the individual instructions.

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.
C_APX	**Near range approximate positioning** (>>> *10.3.6 "Near range approximate positioning" Page 457*) Approximation starts, at the earliest, when the distance to the end point falls below the value of $APO.CDIS. By contrast with C_DIS, $APO.CDIS is set here to considerably lower values. Recommended setting: between 1 and 3 mm.

When using these parameters, Base and Tool must be defined in the approximated block, otherwise the message *{X,Y,Z,A,B,C} BASE not programmed* or *{X,Y,Z,A,B,C} TOOL not programmed* is generated.

PTP, PTP_REL

The parameter can be C_PTP, C_DIS or C_APX for **PTP-PTP** approximation. If a second parameter is specified, the robot controller ignores it.

In some cases, 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.

Combinations for PTP-CP approximation:

1st parameter → / 2nd parameter ↓	C_PTP	C_DIS
(without)	Possible	Possible
C_DIS		Not possible!
C_VEL		Possible
C_ORI		Possible

C_APX can only be used on its own.

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 can be C_DIS, C_VEL, C_ORI or C_APX for **CP-CP** approximation. If a second parameter is specified, the robot controller ignores it.

In some cases, 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.

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	Not possible!	Possible	Possible

C_APX can only be used on its own.

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 to be used in the WITH line of spline commands" Page 577*)
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. (>>> *11.12.3 "Limits for functions in the KRL trigger" Page 645*)
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 to be used in the WITH line of spline commands" Page 577*)
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. (>>> *11.12.3 "Limits for functions in the KRL trigger" Page 645*)
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 to be used in the WITH line of spline commands" Page 577*)
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. (>>> *11.12.3 "Limits for functions in the KRL trigger" Page 645*)
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

```
SPLINE WITH $VEL = SVEL_CP(2.0, , LCPDAT2), $TOOL =
STOOL2(FS2), $BASE = SBASE(FS2.BASE_NO), $IPO_MODE =
SIPO_MODE(FS2.IPO_FRAME), $LOAD = SLOAD(FS2.TOOL_NO), $ACC
= SACC_CP(LCPDAT2), $APO = SAPO(LCPDAT2), $ORI_TYPE =
SORI_TYP(LCPDAT2), $CIRC_TYPE = SCIRC_TYP(LCPDAT2), $JERK =
SJERK(LCPDAT2), $COLLMON_TOL_PRO[1] = USE_CM_PRO_VALUES(1)
SLIN {X 1000, Z -1000} WITH $ACC={CP 5.0,ORI1 500.0,ORI2
500.0}

ENDSPLINE C_Spl
```

```
As individual block:
SLIN {X 1000, Z -1000} 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 443*)
SysVar	(>>> *11.6.11 "System variables to be used in the WITH line of spline commands" Page 577*)
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. (>>> *11.12.3 "Limits for functions in the KRL trigger" Page 645*)
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 to be used in the WITH line of spline commands" Page 577*)
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. (>>> *11.12.3 "Limits for functions in the KRL trigger" Page 645*)

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 to be used in the WITH line of spline commands" Page 577*)
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. (>>> *11.12.3 "Limits for functions in the KRL trigger" Page 645*)
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

```
SPLINE WITH $VEL = SVEL_CP(2.0, , LCPDAT2), $TOOL =
STOOL2(FS2), $BASE = SBASE(FS2.BASE_NO), $IPO_MODE =
SIPO_MODE(FS2.IPO_FRAME), $LOAD = SLOAD(FS2.TOOL_NO), $ACC
= SACC_CP(LCPDAT2), $APO = SAPO(LCPDAT2), $ORI_TYPE =
SORI_TYP(LCPDAT2), $CIRC_TYPE = SCIRC_TYP(LCPDAT2), $JERK =
SJERK(LCPDAT2), $COLLMON_TOL_PRO[1] = USE_CM_PRO_VALUES(1)
```

```
SPTP {A1 90, A2 -45} WITH $ACC={CP 5.0,ORI1 500.0,ORI2
500.0}

ENDSPLINE C_Spl

As individual block:
SPTP {A1 90, A2 -45} 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 564*)

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 to be used in the WITH line of spline commands" Page 577*)
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. (>>> *11.12.3 "Limits for functions in the KRL trigger" Page 645*)

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

```
SPLINE WITH $VEL = SVEL_CP(2.0, , LCPDAT2), $TOOL =
STOOL2(FS2), $BASE = SBASE(FS2.BASE_NO), $IPO_MODE =
SIPO_MODE(FS2.IPO_FRAME), $LOAD = SLOAD(FS2.TOOL_NO), $ACC
= SACC_CP(LCPDAT2), $APO = SAPO(LCPDAT2), $ORI_TYPE =
SORI_TYP(LCPDAT2), $CIRC_TYPE = SCIRC_TYP(LCPDAT2), $JERK =
SJERK(LCPDAT2), $COLLMON_TOL_PRO[1] = USE_CM_PRO_VALUES(1)
SPL XP3
SPL_REL {X 10, Z 10}
ENDSPLINE C_Spl
```

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 564*)

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 443*)
SysVar	(>>> *11.6.11 "System variables to be used in the WITH line of spline commands" Page 577*)
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. (>>> *11.12.3 "Limits for functions in the KRL trigger" Page 645*)
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

```
SCIRC_REL {x 10, y 10}, {x 20, y 20}, CA 200 C_SPL
```

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 564*)

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 to be used in the WITH line of spline commands" Page 577*)
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. (>>> *11.12.3 "Limits for functions in the KRL trigger" Page 645*)
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

```
SPLINE WITH $VEL = SVEL_CP(2.0, , LCPDAT2), $TOOL =
STOOL2(FS2), $BASE = SBASE(FS2.BASE_NO), $IPO_MODE =
SIPO_MODE(FS2.IPO_FRAME), $LOAD = SLOAD(FS2.TOOL_NO), $ACC
= SACC_CP(LCPDAT2), $APO = SAPO(LCPDAT2), $ORI_TYPE =
SORI_TYP(LCPDAT2), $CIRC_TYPE = SCIRC_TYP(LCPDAT2), $JERK =
SJERK(LCPDAT2), $COLLMON_TOL_PRO[1] = USE_CM_PRO_VALUES(1)
SPL XP3
SPL_REL {X 10, Z 10}
ENDSPLINE C_Spl
```

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 564*)

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 to be used in the WITH line of spline commands" Page 577*)
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. (>>> *11.12.3 "Limits for functions in the KRL trigger" Page 645*)

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	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

```
SPLINE WITH $VEL = SVEL_CP(2.0, , LCPDAT2), $TOOL =
STOOL2(FS2), $BASE = SBASE(FS2.BASE_NO), $IPO_MODE =
SIPO_MODE(FS2.IPO_FRAME), $LOAD = SLOAD(FS2.TOOL_NO), $ACC
= SACC_CP(LCPDAT2), $APO = SAPO(LCPDAT2), $ORI_TYPE =
SORI_TYP(LCPDAT2), $CIRC_TYPE = SCIRC_TYP(LCPDAT2), $JERK =
SJERK(LCPDAT2), $COLLMON_TOL_PRO[1] = USE_CM_PRO_VALUES(1)
SPL XP3
SPL_REL {X 10, Z 10}
ENDSPLINE C_Spl
```

11.6.11 System variables to be used in the WITH line of spline commands

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
$EX_AX_IGNORE (>>> *11.6.15 "$EX_AX_IGNORE" Page 589*)
$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.
(>>> *11.12.3 "Limits for functions in the KRL trigger" Page 645*)

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 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 a CP spline block

Description

CONST_VEL is used to define constant velocity ranges. In a constant velocity range, the robot constantly maintains the programmed velocity where possible.

- CONST_VEL can only be used in CP spline blocks.
- A CP spline block can contain either 1 CONST_VEL or 1 TIME_BLOCK, but not both.
- There must be at least 1 spline segment between CONST_VEL END and ENDSPLINE.

A constant velocity range cannot extend beyond the spline block. There is no lower limit to the size of a constant velocity 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.

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. (>>> *11.12.2.2 "Reference point for homogenous approximate positioning" Page 642*)
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. (>>> *11.6.13.2 "Maximum limits of a constant velocity range" Page 584*)

The value for *Offset* can be assigned using a function. The same restrictions apply as for the functions in the trigger.
(>>> *11.12.3 "Limits for functions in the KRL trigger" Page 645*)

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 Block selection to a constant velocity range

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.

11.6.13.2 Maximum limits of a constant velocity range

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.

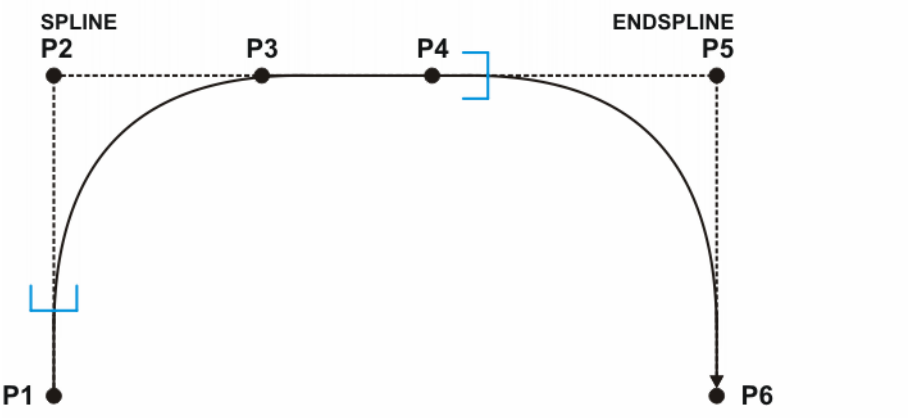


Fig. 11-1: Maximum limits for approximated SPLINE/ENDSPLINE

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.

11.6.13.3 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)	• Message appears in T1/T2 as a notification message.
TRUE	• Message appears in T1/T2 as an acknowledgement message. Acknowledging the message causes the message to be displayed again if the underlying problem has not been eliminated. The message cannot be permanently acknowledged until the programmed velocity has been adapted to the maximum possible constant velocity.

In Automatic operating modes, the messages are always notification messages. They are filtered out by the smartHMI by default and not displayed.

$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 display 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.

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 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}).*

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. (>>> *11.12.2.1 "Reference point for approximate positioning – overview" Page 642*)
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 639*)
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.
(>>> *11.12.3 "Limits for functions in the KRL trigger" Page 645*)

11.6.14.1 Stop condition: example and braking characteristics

Example

The indentations are not present by default and have been inserted here for greater clarity.

The example illustrates programming with inline forms. The principle is the same as that for programming with KRL.

```
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. 11-2: 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. 11-3: 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**.

Situation at BP	Behavior of the robot
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.

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.

11.6.16 $STRIKE_OUT_DEFAULT: preventing velocity drops in the CP spline block

Description

$STRIKE_OUT_DEFAULT can be used to activate the improved geometry planning for CP spline blocks.

Previous behavior:

- If the orientation or an external axis varies greatly in an SLIN or SCIRC segment in the spline block, this may cause drops in velocity or even stops at the transition to an SPL segment.

Behavior with the improved geometry planning:

- The orientations or external axes have a longer effect extending into the adjacent SPL segments. This results in smoother orientation and external axis sequences. Drops in the velocity profile are greatly mitigated and in some cases are completely eliminated.
- The contour of the TCP in Cartesian space (XYZ) is not affected by the adaptations and remains unchanged.



It is advisable to use the improved geometry planning and to retain the preset ABSOLUTE and RELATIVE values unchanged.

Properties

Variable type	Configuration variable
Declaration location	...\STEU\Mada\$custom.dat
Writability	In …\STEU\Mada\$custom.dat

Advance run stop while reading	No advance run stop

Declaration

```
STRIKE_OUT_STRUC $STRIKE_OUT_DEFAULT
```

Explanation

Element	Description
$STRIKE_OUT_DEFAULT	Data type: STRIKE_OUT_STRUC Structure with the settings for improved geometry planning for CP spline blocks

The improved geometry planning is enabled by default in the following case:

- When newly installing a version of the System Software in which $STRIKE_OUT_DEFAULT is available
  $STRIKE_OUT_DEFAULT.MODE = #ON

The improved geometry planning is disabled by default in the following case:

- When upgrading from a version without $STRIKE_OUT_DEFAULT to a version with it
  $STRIKE_OUT_DEFAULT.MODE = #OFF

Data type STRIKE_OUT_STRUC

```
STRUC STRIKE_OUT_STRUC SW_ONOFF MODE, STRIKE_OUT_COMPO-
NENTS ABSOLUTE, RELATIVE
```

Element	Description
MODE	Data type: SW_ONOFF Activation/deactivation of improved geometry planning • #ON: Activated • #OFF: Deactivated
ABSOLUTE	Data type: STRIKE_OUT_COMPONENTS Maximum strike-out angle or distance
RELATIVE	Data type: STRIKE_OUT_COMPONENTS Maximum relative percentage value The percentage values refer to the distances between the start and end points of linearly parameterized orientations and external axes.

11.7 $VEL_APPL: changing the velocity of a CP motion in the main run

Description

$VEL_APPL can be used to modify the process velocity in the main run (i.e. during motion execution). The originally planned profile represents the upper limit and cannot be exceeded.

Field of application:

$VEL_APPL is particularly suitable for processes in which the required velocity only becomes known during program execution. Example: Sensor processes in which the velocity must be varied depending on the circumstances detected by the sensor.

In such processes, the range within which the velocity will lie overall is generally known. Recommendation:

- Define the maximum required process velocity using $VEL.CP.
- Define the lower velocity currently required in the main run using $VEL_APPL.CP.

Properties:

- $VEL_APPL can be applied to all CP motions (LIN, CIRC, CP spline blocks, SLIN, SCIRC).
- $VEL_APPL does not modify the acceleration and braking ramps.
- $VEL_APPL is reset to the default values in the following cases:

  Cold start, **I/O drivers** > **Reconfigure**, loading the machine data, selecting a program, resetting a program, deselecting a program

Constraints:

$VEL_APPL is not effective for BCO runs.

$VEL_APPL cannot be used in the WITH line of a spline command.

With KUKA.RoboTeam:

- $VEL_APPL cannot be used if the synchronization type MotionSync is active.
- $VEL_APPL cannot be used for motions on the LK base coordinate system.

Writability

The variable can be modified via the variable display and via the program.

Writing to the variable in the robot program triggers an advance run stop.

Writing in interrupt programs or via triggers and in the submit program does not trigger an advance run stop.

Syntax

```
$VEL_APPL.MODE = status
$VEL_APPL.CP = velocity
```

Explanation of the syntax

Element	Description
status	Type: ENUM • #OFF (default) • #ON
velocity	Type: REAL; unit: m/s • **0 … 3.0** Default: **0.0** **Note:** The originally planned profile represents the upper limit and cannot be exceeded. If $VEL_APPL.CP is greater than $VEL.CP, the controller takes $VEL.CP.

Example: simple velocity reduction

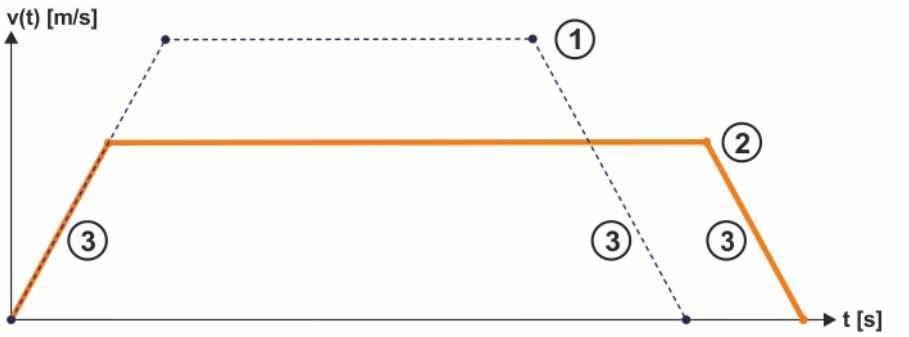


Fig. 11-4: Simple velocity reduction

Item	Description
1	Original velocity profile
2	Profile at reduced velocity
3	The acceleration and braking ramps of the new profile are always identical to the original ramps.

Example: multiple changes in velocity

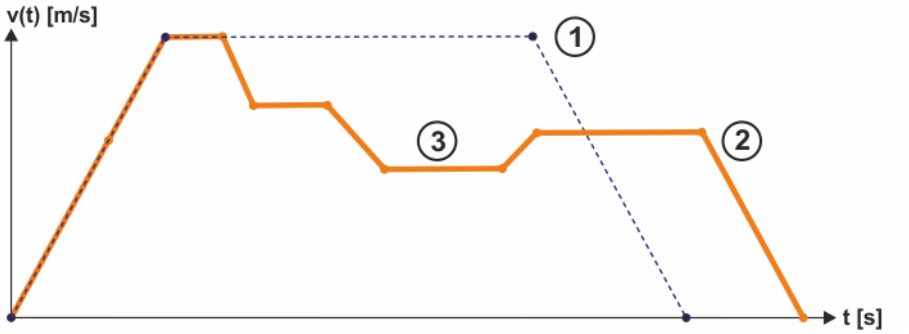


Fig. 11-5: Multiple change in velocity

Item	Description
1	Original velocity profile
2	Profile at reduced velocity
3	Multiple changes in velocity are possible.

Example: Change in velocity in the original profile

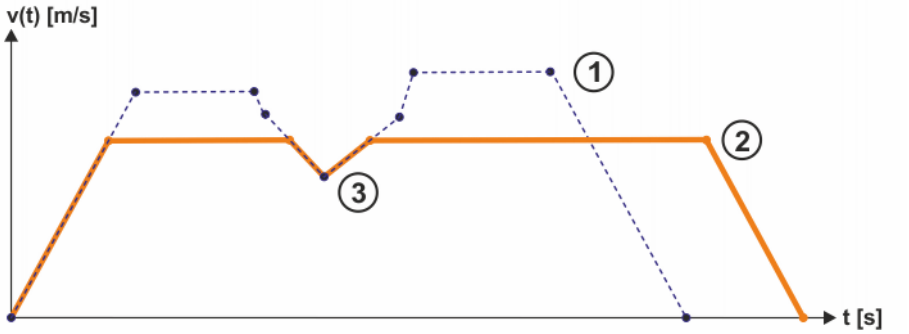


Fig. 11-6: Change in velocity in the original profile

Item	Description
1	Original velocity profile
2	Profile at reduced velocity
3	Changes in velocity in the original profile only have an effect in the new profile where they lie below the limit set by $VEL_APPL.CP.

11.8 Program execution control

11.8.1 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.

11.8.2 EXIT: exit a loop

Description

Exit from a loop. The program is then continued after the loop. EXIT may be used in any loop type.

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
```

11.8.3 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.

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.

Element	Description
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
```

11.8.4 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
```

11.8.5 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
```

11.8.6 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
```

11.8.7 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.

Syntax

```
LOOP
Statements
ENDLOOP
```

Example

```
DECL INT A
A=0
LOOP
 A=A+1
 IF A==10 THEN
  EXIT
 ENDIF
 WAIT SEC 1
ENDLOOP
```

11.8.8 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.

ON_ERROR_PROCEED implicitly deletes the information from $ERR in the current interpreter and at the current level. (>>> *11.8.8.3 "$ERR" Page 601*)

WARNING

Danger to life and limb due to change in program sequence
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. Death, severe injuries or damage to property may result.

- Match the layout of the system to the potential sequence variants. The potential variants must be induced and tested during the program tests.

Constraints

ON_ERROR_PROCEED has no impact on the following instructions.

Motions:

- All spline motions
- MOVE_EMI

Control structures:

- FOR … ENDFOR
- GOTO
- IF … ELSE … ENDIF
- LOOP … ENDLOOP
- REPEAT … UNTIL
- SWITCH … CASE … ENDSWITCH
- WHILE … ENDWHILE

Other: ASYPTP; ASYSTOP; ASYCONT; ASYCANCEL

-

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

(>>> *11.8.8.5 "Program examples: ON_ERROR_PROCEED with ERR_RAISE" Page 605*)

11.8.8.1 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}*

11.8.8.2 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.

Approximate positioning

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 reported 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 transition to the following block by way of approximate positioning.

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.

11.8.8.3 $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.

$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 dif-

ferent 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.

(>>> *11.8.8 "ON_ERROR_PROCEED: skipping an instruction that causes an interpreter error" Page 599*)

Writability

The system variable is write-protected.

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 via the variable display, 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 via the variable display, 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.

11.8.8.4 ERR_RAISE: subsequently generate suppressed message

Description

A message that was suppressed with ON_ERROR_PROCEED can be output by ERR_RAISE subsequently.

ERR_RAISE can only process the $ERR or a variable derived from $ERR as an OUT parameter.

(>>> *11.8.8 "ON_ERROR_PROCEED: skipping an instruction that causes an interpreter error" Page 599*)

(>>> *11.8.8.3 "$ERR" Page 601*)

Syntax

`ERR_RAISE` (`$ERR` | *out_var*)

Explanation of the syntax

Element	Description
out_var	Type: Error_T Variable derived from $ERR as an OUT parameter

Example

(>>> *11.8.8.5 "Program examples: ON_ERROR_PROCEED with ERR_RAISE" Page 605*)

11.8.8.5 Program examples: ON_ERROR_PROCEED with ERR_RAISE

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.

Line	Description
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.

11.8.9 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.

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
```

11.8.10 SWITCH ... CASE ... ENDSWITCH: programming a multiple branch

Description

Executes 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 executed.

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
```

11.8.11 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
```

11.8.12 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
```

11.8.13 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.

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 the last execution of the loop, `A` has the value 100.

```
DECL INT A
A=0
```

```
WHILE A<100
A = A+1
ENDWHILE
```

Example 2

The loop is executed as long as `$IN[1]` is true.

```
DECL INT A
A=0

WHILE $IN[1]=TRUE
A = A+1
ENDWHILE
```

11.9 Inputs/outputs

11.9.1 ANIN ON / ANIN OFF: 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.

Element	Description
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
```

11.9.2 ANOUT ON / ANOUT OFF: 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.
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
```

11.9.3 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:

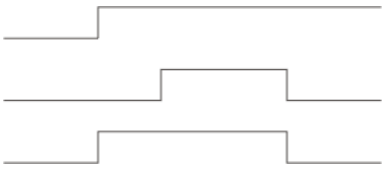


Fig. 11-7: $OUT+PULSE, example 1

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:

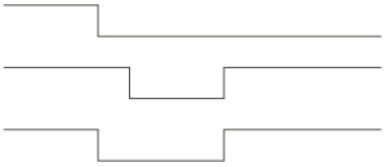


Fig. 11-8: $OUT+PULSE, example 2

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:

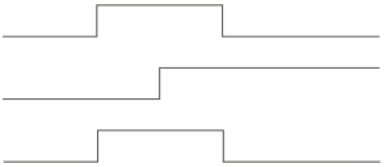


Fig. 11-9: PULSE+$OUT, example 1

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:

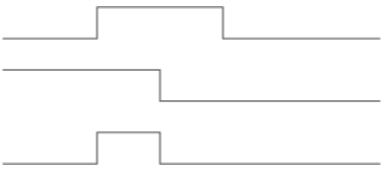


Fig. 11-10: PULSE+$OUT, example 2

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:

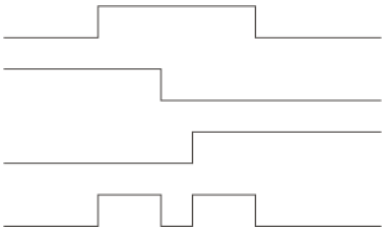


Fig. 11-11: PULSE+$OUT, example 3

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):

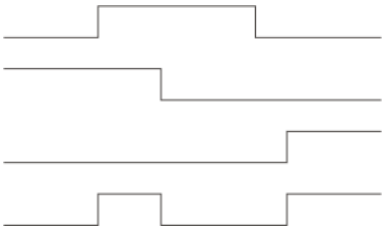


Fig. 11-12: PULSE+$OUT, example 4

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:

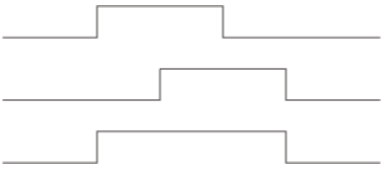


Fig. 11-13: PULSE+PULSE, example 1

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:

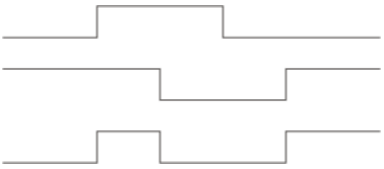


Fig. 11-14: PULSE+PULSE, example 2

```
PULSE($OUT[50],TRUE,3.0)
PULSE($OUT[50],FALSE,1.0)
```

Actual pulse characteristic at output 50:

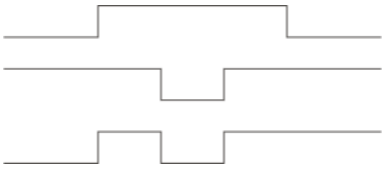


Fig. 11-15: PULSE+PULSE, example 3

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:

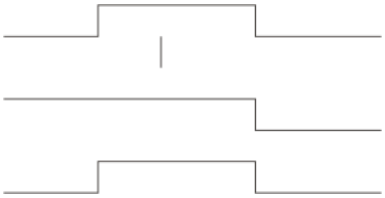


Fig. 11-16: PULSE+END, example

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:

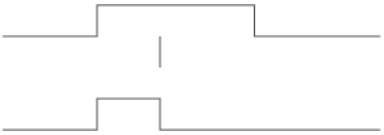


Fig. 11-17: PULSE+RESET, example

11.9.4 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 549*)
Signal name	Any name
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'
```

11.10 Subprograms and functions

11.10.1 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()
```

11.10.2 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
```

11.10.3 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 (>>> *11.10.5 "Transferring parameters to a subprogram or function" Page 621*)
`IN | OUT`	If a value is transferred in the function: type of transfer
Function value	(>>> *11.10.4 "RETURN: jump back to the calling program" Page 620*)

Example

(>>> *11.10.4 "RETURN: jump back to the calling program" Page 620*)

11.10.4 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
```

11.10.5 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.
(>>> *11.10.6 "Transferring a parameter to a different data type" Page 625*)

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).

11.10.6 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.

11.11 Interrupt programming

11.11.1 INTERRUPT … DECL … WHEN … DO … : Declare interrupt

Description

In the case of a defined event, e.g. an edge change of 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.

No interruption of current motions

Current and prepared robot motions are not generally interrupted by an interrupt. Parallel to the interrupt program, the robot continues to execute all motions that have already been prepared in the main program.

If the interrupt program is completely executed during this time, the main program is resumed without any pause between motions, i.e. without the processing time being lengthened.

Maximum number / BRAKE

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.
(>>> *11.11.2 "INTERRUPT ON/OFF: activating or deactivating an interrupt" Page 628*)

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. This means: 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. (>>> *11.11.4 "BRAKE: stopping the robot from an interrupt program" Page 631*)
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.

Element	Description
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 All interrupts are edge-triggered and react to events in the form of edge changes.
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
```

11.11.2 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
```

11.11.3 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 630*)

- 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 630*)

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
```

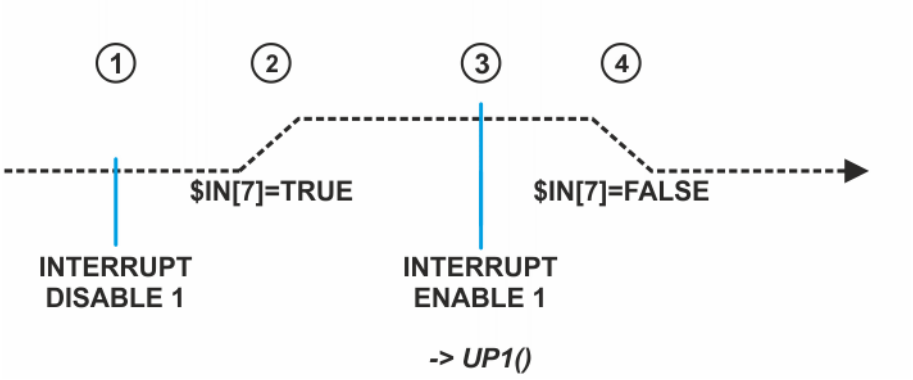


Fig. 11-18: Example 1

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
```

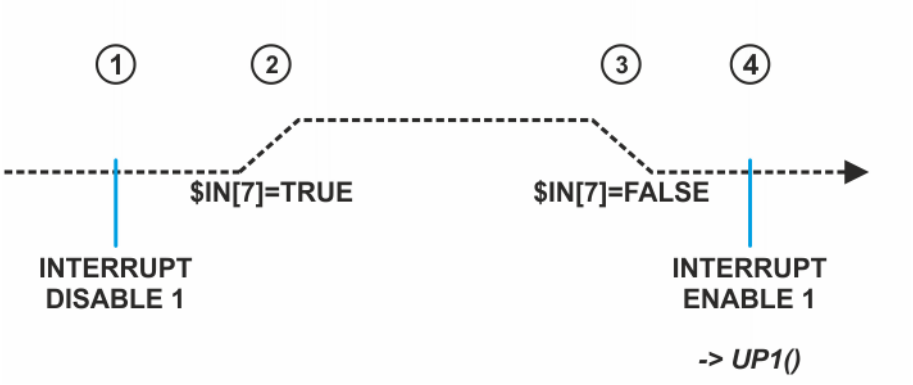


Fig. 11-19: Example 2

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.

11.11.4 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 initiating 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: (>>> *11.11.2 "INTERRUPT ON/OFF: activating or deactivating an interrupt" Page 628*)

11.11.5 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.

11.12 Path-related switching actions (=Trigger)

11.12.1 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 635*)
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 636*) If *Time* calls a function, the function is subject to constraints. (>>> *11.12.3 "Limits for functions in the KRL trigger" Page 645*)
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.

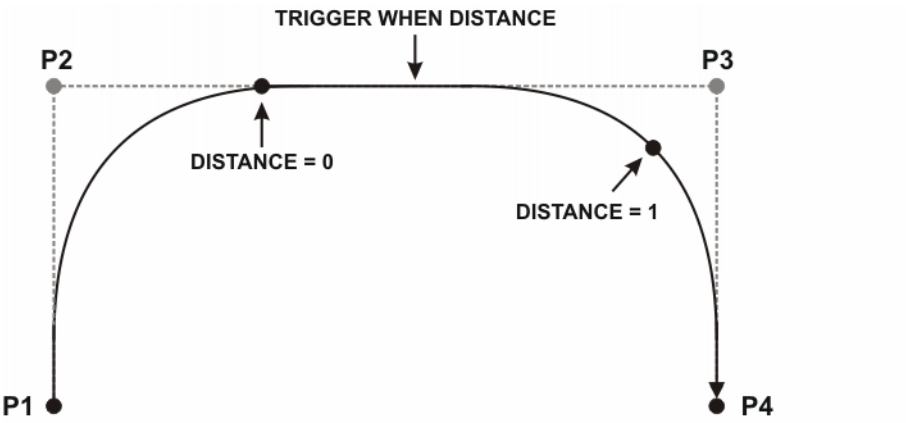


Fig. 11-20: TRIGGER WHEN DISTANCE reference point for approximate positioning

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. 11-21*)**:**

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

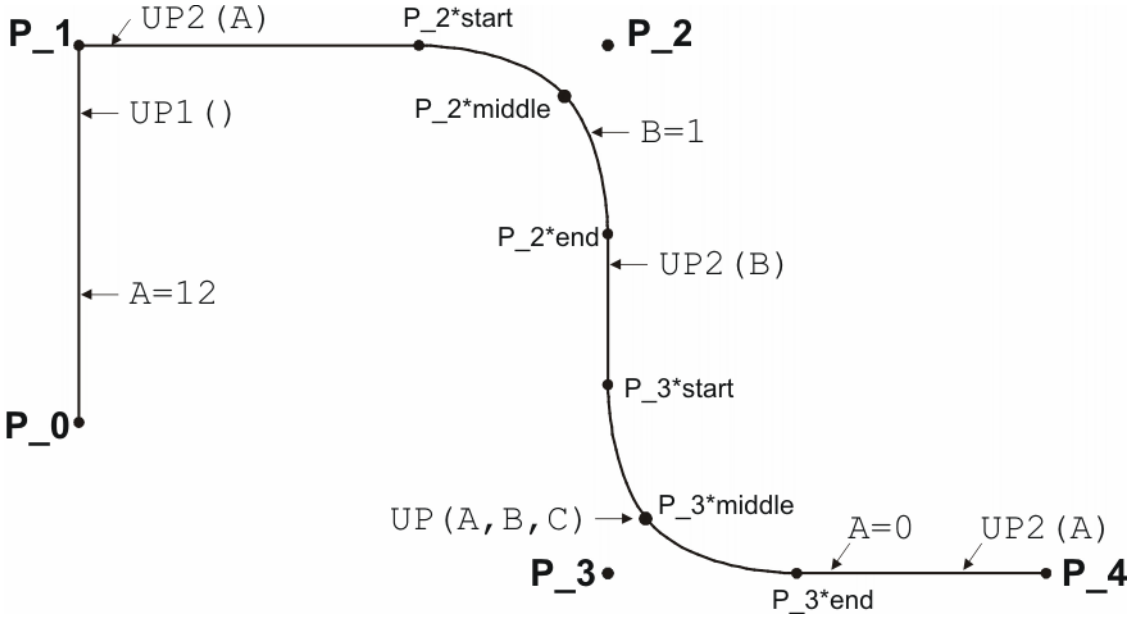


Fig. 11-21: Example of TRIGGER WHEN DISTANCE

11.12.2 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.

(>>> *11.12.3 "Limits for functions in the KRL trigger" Page 645*)

Explanation of the syntax

Element	Description
ONSTART	Reference point of the trigger • With ONSTART: Start point • Without ONSTART: End point (>>> *11.12.2.1 "Reference point for approximate positioning – overview" Page 642*)
Path	Type: REAL; variable, constant or function; unit: mm (except in the case of PTP splines without unit) Shift in space, starting from 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 639*)
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 639*)
Instruction	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. (>>> *11.12.2.3 "Reference point for mixed approximate positioning (spline)" Page 644*) • 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.

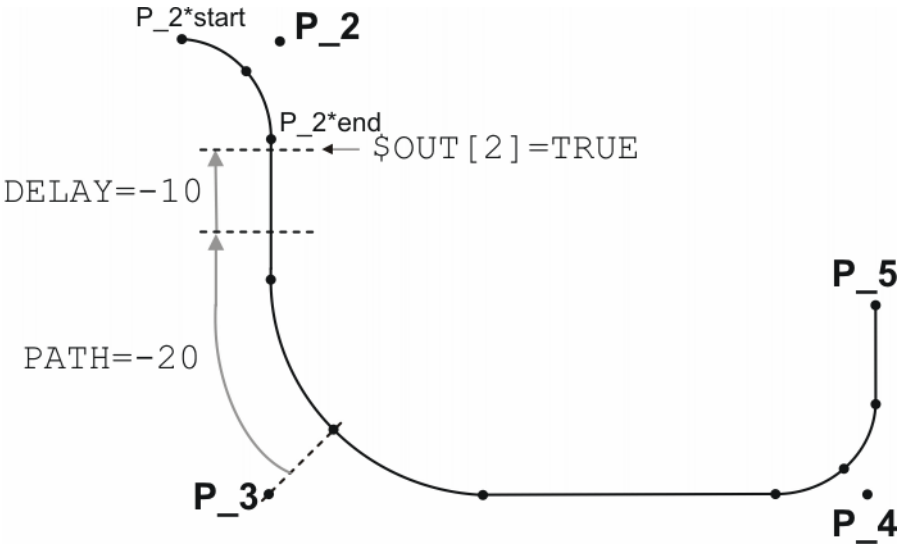


Fig. 11-22: Example of TRIGGER WHEN PATH

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.

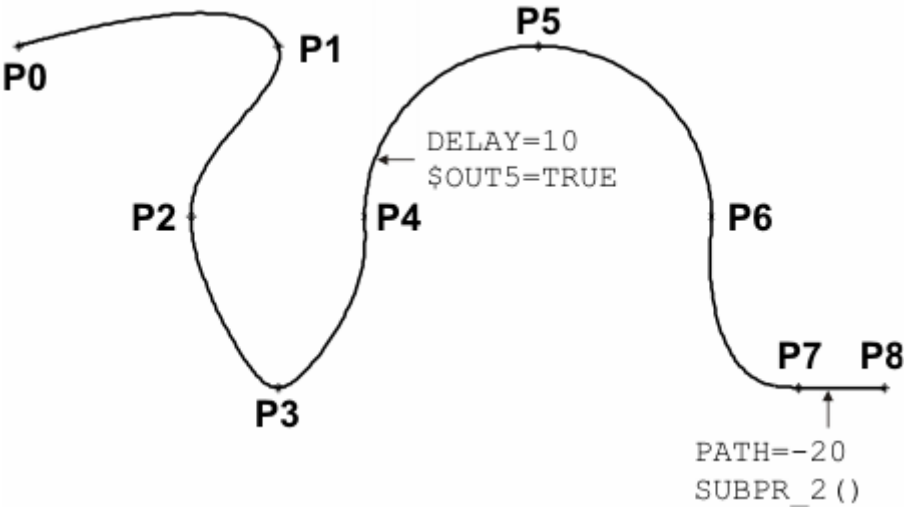


Fig. 11-23: Example of TRIGGER WHEN PATH (for spline)

11.12.2.1 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 homogeneous or mixed approximate positioning is being carried out.

Homogeneous

Homogeneous 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

Any spline motion can be a spline block or an individual instruction.

(>>> *11.12.2.2 "Reference point for homogenous approximate positioning" Page 642*)

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

  Any spline motion can be a spline block or an individual instruction.

  (>>> *11.12.2.3 "Reference point for mixed approximate positioning (spline)" Page 644*)
- From a PTP spline motion to a LIN or CIRC spline motion or vice versa

  (>>> *11.12.2.4 "Reference point for mixed approximate positioning (LIN/CIRC/PTP)" Page 645*)

11.12.2.2 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 $P_{StartApprox} \rightarrow P3$ is the same as $P_{StartApprox} \rightarrow P3'_{Trigger\ 1}$.



Fig. 11-24: Trigger 1: Reference point for homogenous approximate positioning

$P_{StartApprox}$	Start of the approximate positioning arc
P3	Reference point for exact positioning
$P3'_{Trigger\ 1}$	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 $P_{EndApprox} \rightarrow P3$ is the same as $P_{EndApprox} \rightarrow P3'_{Trigger\ 2}$.

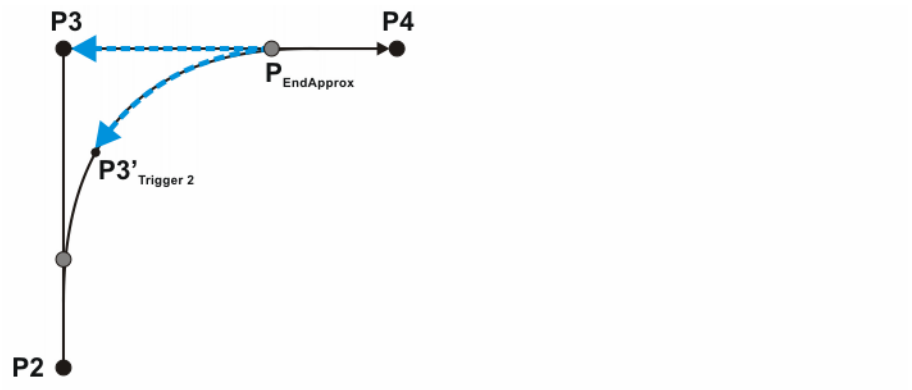


Fig. 11-25: Trigger 2: Reference point for homogenous approximate positioning

$P_{EndApprox}$	End of the approximate positioning arc
P3	Reference point for exact positioning
$P3'_{Trigger\ 2}$	Reference point for approximate positioning

11.12.2.3 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 643*)

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 $P_{StartApprox} \rightarrow P3'_{Trigger\ 1}$ is generally shorter than $P_{StartApprox} \rightarrow P3$.

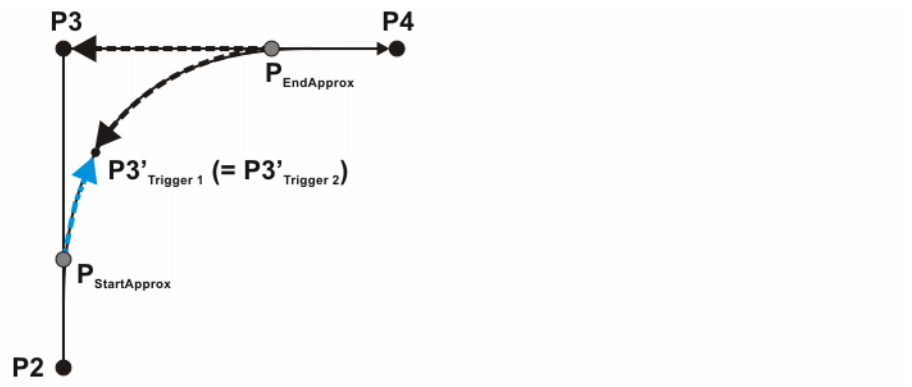


Fig. 11-26: Trigger 1: Reference point for mixed approximate positioning

$P_{StartApprox}$	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 **$P_{StartApprox}$** → **$P3'_{Trigger\ 1}$**

11.12.2.4 Reference point for mixed approximate positioning (LIN/CIRC/PTP)

Start point approximated (for trigger with ONSTART)

The PATH trigger is only possible for CP motions. As a result, this is an approximate positioning from PTP to CP in the case of “with ONSTART”.

Reference point: The reference point is at the end of the approximate positioning arc.

End point approximated (for trigger without ONSTART)

The PATH trigger is only possible for CP motions. As a result, this is an approximate positioning from CP to PTP in the case of “without ONSTART”.

Reference point: The reference point is located where the end point of the CP motion would lie in the event of exact positioning.

11.12.3 Limits for functions in the KRL trigger

In the case of trigger programming in KRL, 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 342*)
- 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

11.12.4 System variables for working with PATH triggers

11.12.4.1 $DIST_NEXT

Description

$DIST_NEXT specifies the length of the path from the current TCP position to the next taught point.

Writability

The system variable is write-protected.

Syntax

Distance = `$DIST_NEXT`

Explanation of the syntax

Element	Description
Distance	Type: REAL Length of the path from the current TCP position to the next taught point Unit: • For CP motions (spline and conventional): mm • For SPTP motions: no unit The value is generally negative. $DIST_NEXT cannot be used for PTP motions. The value is always zero in this case.

Programming a PATH trigger with the aid of $DIST_NEXT

$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.

(>>> *11.12.4.3 "Response when direction changes" Page 647*)

11.12.4.2 $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.

Writability

The system variable is write-protected.

Syntax

Distance = `$DIST_LAST`

Explanation of the syntax

Element	Description
Distance	Type: REAL Length of the path from the current TCP position to the previous taught point Unit: • For CP motions (spline and conventional): mm • For SPTP motions: no unit The value is generally positive. $DIST_LAST cannot be used for PTP motions. The value is always zero in this case.

Programming a PATH trigger with the aid of $DIST_LAST

$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.

(>>> *11.12.4.3 "Response when direction changes" Page 647*)

11.12.4.3 Response when direction changes

Description

The value of $DIST_LAST is generally positive. The value of $DIST_NEXT is generally negative.

Spline:

If the PATH trigger relates to a spline motion, then the sign preceding $DIST_NEXT and $DIST_LAST does not change when there is a change of direction.

Conventional CP motion (other than PTP):

If the PATH trigger relates to LIN, CIRC, LIN_REL or CIRC_REL, then the following applies:

- The current position becomes the starting point when the trigger is added. Consequently, $DIST_LAST = 0.
- The current position also becomes the starting point when there is a change of direction using one of the start keys.
- If the Start backwards key is pressed, then the programmed starting point becomes the target point. i.e. $DIST_NEXT displays the value displayed previously by $DIST_LAST, only this time with a negative preceding sign.

Example

```
...
LIN P5
LIN P6
...
```

1. The distance from P5 to P6 is 81 mm. The TCP has covered 24 mm in the forwards direction and is now stationary. Result:
   - $DIST_LAST = 24
   - $DIST_NEXT = -(81 - 24) = -57
2. For diagnosis purposes, we set the override to zero: this means that the TCP does not move and it is now easier to monitor the values.
3. If you add a PATH trigger between LIN P5 and LIN P6 and press the Start forwards key, the result is:
   - $DIST_LAST = 0
   - $DIST_NEXT = -57

   Reason: the current position became the starting point when the trigger was added.
4. If you now press Start backwards (it does not matter if the abovementioned trigger was added or not), the result is:
   - $DIST_LAST = 0
   - $DIST_NEXT = -24

   Reason: pressing Start backwards causes the programmed starting point to become the target point.
5. If you now press Start forwards, the result is:
   - $DIST_LAST = 0
   - $DIST_NEXT = -57

11.13 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

11.14 Operators

In each operation, the compiler checks the legitimacy of the operands.

11.14.1 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.

Further information about the order in which the operators are processed:

- (>>> *11.14.6 "Priority of the operators" Page 657*)

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
```

11.14.2 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.		

11.14.2.1 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}

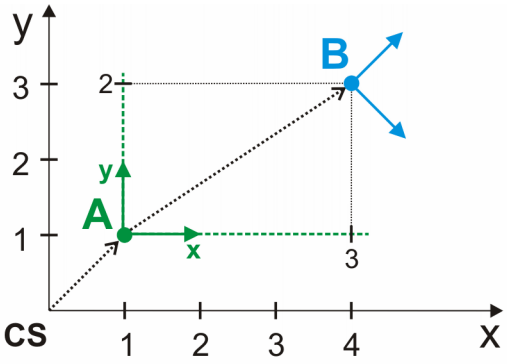


Fig. 11-27: R = A : B

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}

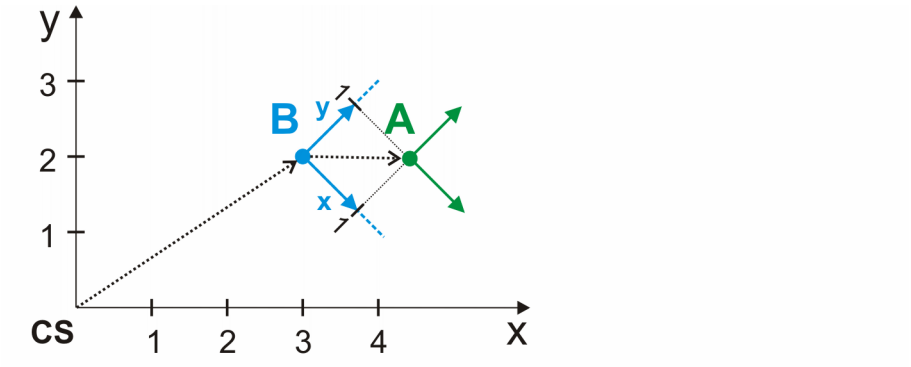


Fig. 11-28: R = B : A

11.14.2.2 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

```
 1  DEF geo_operator( )
 2  DECL AXIS home
 3  DECL FRAME ref_pos_x, ref_pos_y, ref_pos_z
 4  DECL FRAME My_BASE[2]
...
 5  INI
 6  home={AXIS: A1 0,A2 -90,A3 90,A4 0,A5 30,A6 0}
 7  $BASE={X 1000,Y 0,Z 1000,A 0,B 0,C 0}
 8  ref_pos_X={X 100,Y 0,Z 0,A 0,B 0,C 0}
 9  ref_pos_Y={X 100,Y 100,Z 0,A 0,B 0,C 0}
10  ref_pos_Z={X 100,Y 100,Z 100,A 0,B 0,C 0}
11  My_BASE[1]={X 200,Y 100,Z 0,A 0,B 0,C 180}
12  My_BASE[2]={X 0,Y 200,Z 250,A 0,B 90,C 0}
...
13  PTP home
14  PTP {X 0,Y 0,Z 0,A 0,B 0,C 0}
15  WAIT SEC 2
16  PTP ref_pos_X
17  PTP ref_pos_Y
18  PTP ref_pos_Z
19  PTP My_BASE[1]
20  WAIT SEC 2
21  PTP My_BASE[1]:ref_pos_X
22  PTP My_BASE[1]:ref_pos_Y
23  PTP My_BASE[1]:ref_pos_Z
24  PTP My_BASE[1]:My_BASE[2]
25  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].

11.14.3 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	
>	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.

Further information about the order in which the operators are processed:
- (>>> *11.14.6 "Priority of the operators" Page 657*)

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
```

11.14.4 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
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.

Further information about the order in which the operators are processed:

- (>>> *11.14.6 "Priority of the operators" Page 657*)

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
...
```

11.14.5 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 results 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 […]”.

Further information about the order in which the operators are processed:
- (>>> *11.14.6 "Priority of the operators" Page 657*)

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. 11-29: 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. 11-30: 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. 11-31: 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. 11-32: 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
```

11.14.6 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.

11.15 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.

Further information about the order in which the operators are processed:

- (>>> *11.14.6 "Priority of the operators" Page 657*)

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
```

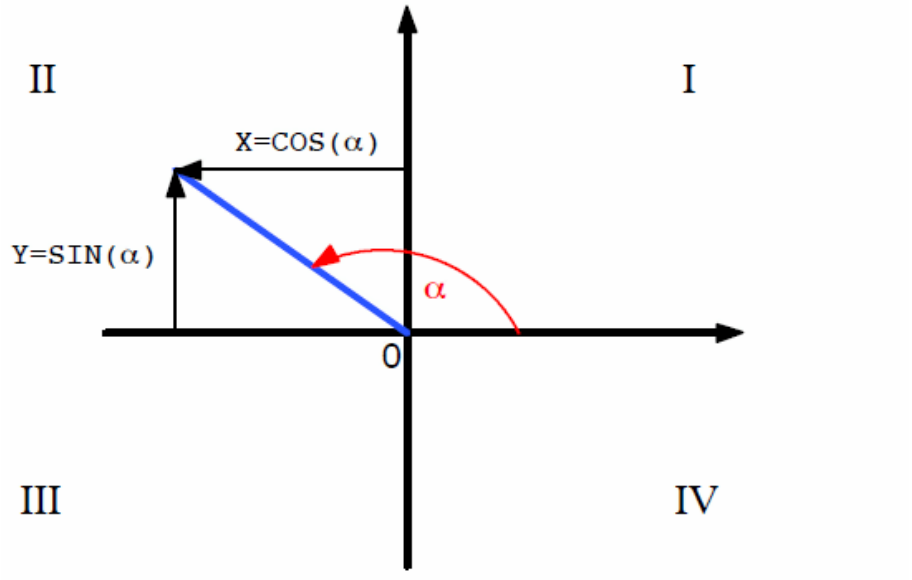


Fig. 11-33: Use of X and Y in the function ATAN(Y,X)

11.16 System functions

11.16.1 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.

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.

Example

In the following example, the following actions are possible or not possible:

Action	Possible?
Move backwards from P5	Yes
Move backwards to P5	No
Move backwards to P6	Yes

```
PTP P3
PTP P4
PTP P5
DELETE_BACKWARD_BUFFER()
PTP P6
PTP P7
```

If, for example, it is desirable for no backward motion to be possible after P3 has been left and before P6 has been reached, the function must be programmed several times:

```
PTP P3
DELETE_BACKWARD_BUFFER()
PTP P4
DELETE_BACKWARD_BUFFER()
PTP P5
DELETE_BACKWARD_BUFFER()
PTP P6
PTP P7
```

11.16.2 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, both in the form of a Robroot kinematic system and as a Base kinematic system.
- The function can be called via the variable display 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 Type of transfer: IN parameter Axis angles of the robot for which the Cartesian position is to be calculated

Element	Description
err_status	Type: INT Type of transfer: OUT parameter The variable must be initialized and transferred when the function is called; it then returns any error codes. Initial value: Setting whether the transferred axis angles are to be checked against software limit switches: • 0: All axis angles are tested. If they are outside the limits of the software limit switches, *err_status* returns an error code. • >0: Axis angles are not checked; a different calculation strategy is used (not recommended). • <0: Invalid transfer value.

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	$TOOL is invalid.
-3	$BASE is invalid.
-2	*err_status* has been initialized with an invalid 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

11.16.3 GET_SIG_INF: reading information about signal declaration

Description

GET_SIG_INF() can be used to read information about a signal declaration – particularly which inputs or outputs are assigned to it.

GET_SIG_INF() can also be used via the variable display.

Syntax

result = `GET_SIG_INF (`*signal_name*`)`

Explanation of the syntax

Element	Description
result	Type: SIGINF Variable for the return value
signal_name	Type: CHAR • Signal name (in quotation marks) • Or variable to which the signal name has been assigned (without quotation marks)

SIGINF is a structure type with the following components:

Component	Value	Meaning
TYP Type: SIGTYP	#IN	Signal is declared for inputs.
	#OUT	Signal is declared for outputs.
	#INV	Signal is not declared or has been set to FALSE.
DTYP Type: SIGDTYP	#SYSI	System signal of type INT
	#SYSBI	System signal of type BOOL
	#USRI	User-defined global signal of type INT
	#USRB	User-defined global signal of type BOOL
	#UNKNOWN	Signal is not declared or has been set to FALSE.
IDX Type: INT	1 … 8192	Number of the input/output For signals that group together several I/Os: the start value
	0	Signal number is invalid or signal has been set to FALSE.
LEN Type: INT	1 … 32	Signal length Always “1” for Boolean signals
	0	Length is invalid or signal has been set to FALSE.

Example

```
 1 DEF SIGTEST ( )
 2 DECL SIGINF S_INF
 3 CHAR S_NAME[24]
 4 S_NAME[] = "$EXT_START"
 5 ...
 6 S_INF = GET_SIG_INF("$MOVE_ENABLE")
 7 IF (S_INF.IDX == 1025) OR (S_INF.IDX == 0) THEN
 8    HALT
 9 ENDIF
10 ...
11 S_INF = GET_SIG_INF(S_NAME[])
12 IF (S_INF.TYP == #INV) OR (S_INF.IDX == 0) THEN
13    HALT
14 ENDIF
15 ...
16 END
```

Variable display

- Main menu **Display** > **Variable** > **Single**

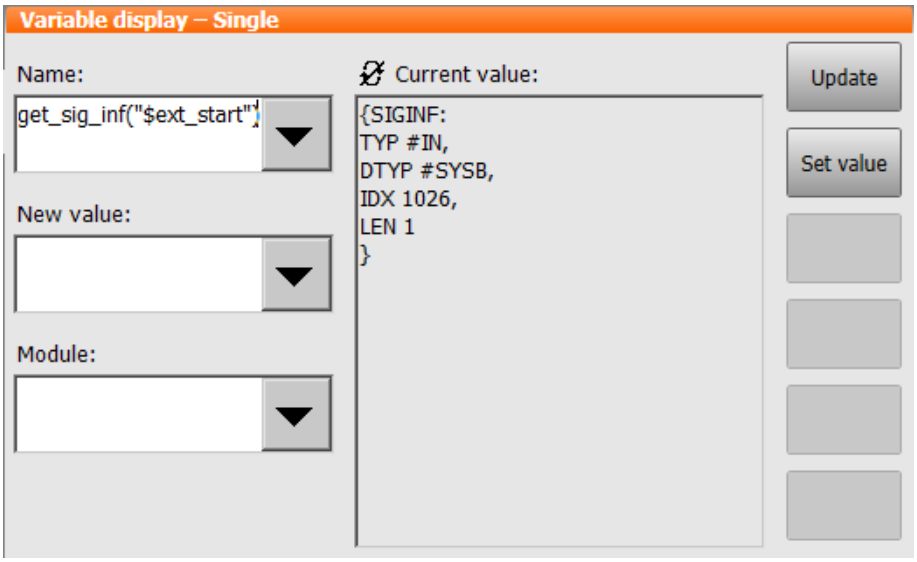


Fig. 11-34: Example: reading out $EXT_START via GET_SIG_INF()

11.16.4 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 (:). (>>> *11.14.2 "Geometric operator" Page 650*)

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.

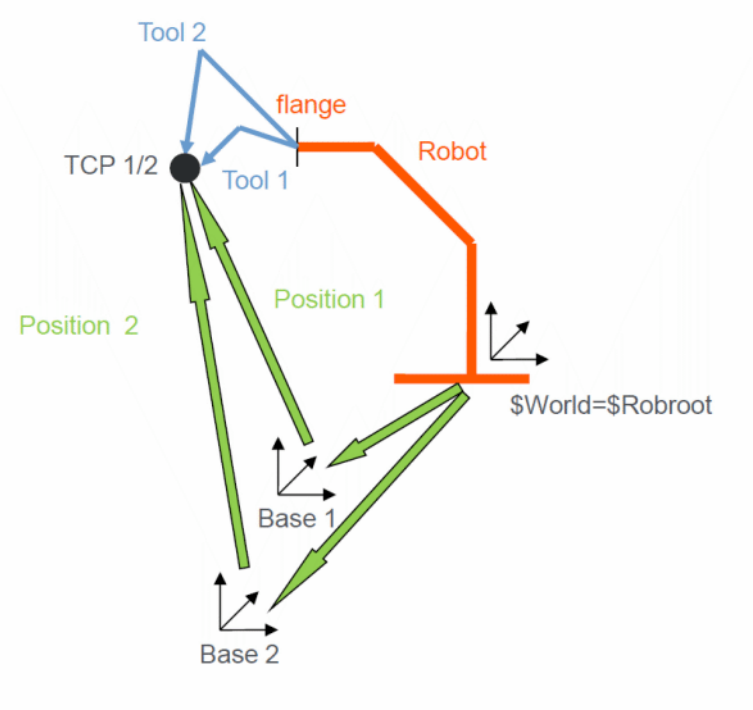


Fig. 11-35: INV_POS example

```
Position2=INV_POS(BASE_DATA[2]):BASE_DATA[1]:Position1:INV_PO
S(TOOL_DATA[1]):TOOL_DATA[2]
```

11.16.5 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 via the variable display 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 Type of transfer: 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 Type of transfer: IN parameter Axis angles of the robot at the start point of the motion
err_status	Type: INT Type of transfer: OUT parameter The variable must be initialized and transferred when the function is called; it then returns any error codes. Initial value: 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 outside the limits of the software limit switches, *err_status* returns an error code. • >0: Axis angles are not checked; a different calculation strategy is used (not recommended). • <0: Invalid transfer value. 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	$TOOL is invalid.
-3	$BASE is invalid.
-2	*err_status* has been initialized with an invalid 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. The code 0 is also returned if the Cartesian end position has a Status and Turn but the start position is outside the software limit switches. The start position will not be evaluated in this case.
1	Software limit switches violated at the start position with the specified axis angles; Cartesian end position has no Status or Turn.
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. However, a software limit switch is violated with the specified Turn.

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 also available as standard on every controller:

Directory	C:\KRC\UTIL\KUEWEG
File	kue_weg.src

11.16.6 ROB_STOP: prevent robot motions

Description

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.

A blockade caused by ROB_STOP is canceled with ROB_STOP_RELEASE. (>>> *11.16.7 "ROB_STOP_RELEASE: permit robot motions again" Page 669*)

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.

ROB_STOP and ROB_STOP_RELEASE can only be used in submit programs.

Message

ROB_STOP triggers the following status message: *Robot stopped by submit*

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 Type of transfer: 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.

Example

```
 1 DECLARATIONS
 2 decl bool my_var
 3 INI
 4
 5 LOOP
 6 ...
 7 my_var = rob_stop (#ramp_down)
 8 ...
 9 rob_stop_release()
10 USER PLC
11 ENDLOOP
12 USER SUBROUTINE
```

Line	Description
7	The robot stops with a ramp stop.
9	The blockade is canceled. Robot motion is resumed.

11.16.7 ROB_STOP_RELEASE: permit robot motions again

Description

ROB_STOP_RELEASE cancels a blockade caused by ROB_STOP. (>>> *11.16.6 "ROB_STOP: prevent robot motions" Page 667*)

If ROB_STOP_RELEASE is called without ROB_STOP being called first, this has no effect.

ROB_STOP() and ROB_STOP_RELEASE() can only be used in submit programs.

Message

ROB_STOP_RELEASE in a Test mode triggers the following acknowledgment message: *Ackn. Robot stopped by submit*

ROB_STOP_RELEASE in an Automatic mode triggers no message.

Syntax

```
ROB_STOP_RELEASE ()
```

Example

```
 1 DECLARATIONS
 2 decl bool my_var
 3 INI
 4
 5 LOOP
 6 ...
 7 my_var = rob_stop (#ramp_down)
 8 ...
 9 rob_stop_release()
10 USER PLC
11 ENDLOOP
12 USER SUBROUTINE
```

Line	Description
7	The robot stops with a ramp stop.
9	The blockade is canceled. Robot motion is resumed.

11.16.8 SET_BRAKE_DELAY: reducing the delay time for applying the brakes

Description

The function SET_BRAKE_DELAY can be used to reduce the delay time for applying the brakes 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.

Delay time for applying the brakes:

The delay time for applying the brakes 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 delay time for applying the brakes is defined in $BRK_DEL (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.

Element	Description
delay	Type: INT; unit: ms Desired delay. Range of values: • **0** … defined general 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

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 delay time is set to 0 ms for all robot axis brakes 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 delay time is set to 0 ms for all robot axis brakes 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 comparison: 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.

11.16.9 TIMER_LIMIT: timer that only runs with an active interpreter

Description

As soon as the interpreter is stopped, the internal function provides a timer that only runs and stops with an active interpreter. The function can only be used in a WAIT FOR statement. (>>> *11.8.11 "WAIT FOR … : waiting until a condition has been met" Page 609*)

Syntax

`WAIT FOR TIMER_LIMIT(`*seconds*`)`

Explanation of the syntax

Element	Description
seconds	Type: INT Number of seconds that the timer counts with an active interpreter. Once the set time has elapsed, the function returns the value TRUE and the WAIT FOR statement is left.

Example

The program waits until the interpreter has been running for 10 seconds. It is not necessary for the interpreter to run for 10 seconds without interruption, as long as it runs for a total of 10 seconds.

```
WAIT FOR TIMER_LIMIT(10)
```

11.16.10 VARSTATE: polling the state of a variable

Description

VARSTATE() can be used to monitor the state of a variable.

The data type of the variable must not exceed a certain degree of complexity: Only the data types BOOL, INT, REAL, CHAR and ENUMs are supported, and these can also be elements of structures or arrays. Structures or ENUMs as components of a structure are not supported, nor is the specification of an integer variable as an array index.

Syntax

result = VARSTATE ("*var_name*")

Explanation of the syntax

Element	Description
result	Type: VAR_STATE Variable for the return value. Possible values: • #DECLARED: Variable is declared, but not initialized. • #INITIALIZED: Variable is initialized. • #UNKNOWN: Variable is unknown/not declared.
var_name	Type: CHAR Type of transfer: IN parameter Name of the variable whose status is to be queried (max. 80 characters)

Example

```
 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.

11.17 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 display.

The functions can be used within IF branches without the return value being explicitly assigned to a variable.

11.17.1 StrTo… : converting a character string to a different data type

11.17.1.1 StrToAXIS: converting character string to AXIS

Description

The function **StrToAXIS()** can be used to convert a character string to a variable of data type AXIS.

Syntax

result = `StrToAXIS(`*string*`,` *value*`)`

Explanation of the syntax

Element	Description
result	Type: BOOL Variable for the return value • TRUE: Conversion successful • FALSE: Conversion not successful
string	Type: CHAR Character string that is to be converted to a different data type. Maximum 256 characters. • If *string* represents an aggregate, the individual components must be separated by commas. • Missing components are not initialized and thus remain without a value. • If there are already components present in the target variable, they are deleted.

Element	Description
value	Type: AXIS Variable for the value after conversion

Examples

```
1 DECL AXIS value
2 DECL BOOL result
3 ...
4 result=StrToAXIS("{A1 90.3}",value)
5 result=StrToAXIS("{E1 100}",value)
6 result=StrToAXIS("{A1 90.3, E1 100}",value)
7 result=StrToAXIS("{E1 100, A1 90.3}",value)
```

Line	Description
4	Conversion is possible, as A1 is a component of AXIS. The return value *result* is TRUE.
5	Conversion is not possible, as E1 is not a component of AXIS. The return value *result* is FALSE.
6	Conversion is possible. The value of E1 is lost. The return value *result* is TRUE.
7	The same result as in line 6

11.17.1.2 StrToE3AXIS: converting character string to E3AXIS

Description

The function **StrToE3AXIS()** can be used to convert a character string to a variable of data type E3AXIS.

Syntax

result = StrToE3AXIS(*string*, *value*)

Explanation of the syntax

Element	Description
result	Type: BOOL Variable for the return value • TRUE: Conversion successful • FALSE: Conversion not successful
string	Type: CHAR Character string that is to be converted to a different data type. Maximum 256 characters. • If *string* represents an aggregate, the individual components must be separated by commas. • Missing components are not initialized and thus remain without a value. • If there are already components present in the target variable, they are deleted.
value	Type: E3AXIS Variable for the value after conversion

Examples

```
1 DECL E3AXIS value
2 DECL BOOL result
3 ...
4 result=StrToE3AXIS("{A1 90.3, E1 100}",value)
5 result=StrToE3AXIS("{E4 100}",value)
6 result=StrToE3AXIS("{A1 90.3, E4 100}",value)
7 result=StrToE3AXIS("{E4 100, A1 90.3}",value)
```

Line	Description
4	Conversion is possible, as A1 and E1 are a component of E3AXIS. The return value *result* is TRUE.
5	Conversion is not possible, as E4 is not a component of E3AXIS. The return value *result* is FALSE.
6	Conversion is possible. The value of E4 is lost. The return value *result* is TRUE.
7	The same result as in line 6

11.17.1.3 StrToE6AXIS: converting character string to E6AXIS

Description

The function **StrToE6AXIS()** can be used to convert a character string to a variable of data type E6AXIS.

Syntax

result = StrToE6AXIS(*string*, *value*)

Explanation of the syntax

Element	Description
result	Type: BOOL Variable for the return value • TRUE: Conversion successful • FALSE: Conversion not successful
string	Type: CHAR Character string that is to be converted to a different data type. Maximum 256 characters. • If *string* represents an aggregate, the individual components must be separated by commas. • Missing components are not initialized and thus remain without a value. • If there are already components present in the target variable, they are deleted.
value	Type: E6AXIS Variable for the value after conversion

Examples

```
1 DECL E6AXIS value
2 DECL BOOL result
```

```
3 ...
4 result=StrToE6AXIS("{A1 90.3, E4 100}",value)
5 result=StrToE6AXIS("{A1 90.3, X 10}",value)
```

Line	Description
4	Conversion is possible, as A1 and E4 are a component of E6AXIS. The return value *result* is TRUE.
5	Conversion is not possible. Even though A1 is a component of E6AXIS, conversion is not possible due to the combination with X (or another non-axis-specific value). The return value *result* is FALSE.

11.17.1.4 StrToFRAME: converting character string to FRAME

Description

The function **StrToFRAME()** can be used to convert a character string to a variable of data type FRAME.

Syntax

result = `StrToFRAME(`*string*`,` *value*`)`

Explanation of the syntax

Element	Description
result	Type: BOOL Variable for the return value • TRUE: conversion successful • FALSE: conversion not successful
string	Type: CHAR Character string that is to be converted to a different data type. Maximum 256 characters. • If *string* represents an aggregate, the individual components must be separated by commas. • Missing components are not initialized and thus remain without a value. • If there are already components present in the target variable, they are deleted.
value	Type: FRAME Variable for the value after conversion

Examples

```
1 DECL FRAME value
2 DECL BOOL result
3 ...
4 result=StrToFRAME("{X 10, Y 50.7, C 45}",value)
5 result=StrToFRAME("{X10}",value)
6 result=StrToFRAME("{E1 100}",value)
7 result=StrToFRAME("{X 10, E1 100}",value)
8 result=StrToFRAME("{E1 100, X 10}",value)
```

Line	Description
4	Conversion is possible. The return value *result* is TRUE. The value after conversion is “X 10, Y 50.7, C 45”.
5	Conversion is not possible. The return value *result* is FALSE. It must be spelled "X 10", i.e. with a space.
6	Conversion is not possible, as E1 is not a component of FRAME. The return value *result* is FALSE.
7	Conversion is possible. The value of E1 is lost. The return value *result* is TRUE. The value after conversion is "X 10".
8	The same result as in line 7

11.17.1.5 StrToPOS: converting character string to POS

Description

The function **StrToPOS()** can be used to convert a character string to a variable of data type POS.

Syntax

result = `StrToPOS(`*string*`,` *value*`)`

Explanation of the syntax

Element	Description
result	Type: BOOL Variable for the return value • TRUE: conversion successful • FALSE: conversion not successful
string	Type: CHAR Character string that is to be converted to a different data type. Maximum 256 characters. • If *string* represents an aggregate, the individual components must be separated by commas. • Missing components are not initialized and thus remain without a value. • If there are already components present in the target variable, they are deleted.
value	Type: POS Variable for the value after conversion

Examples

```
1 DECL POS value
2 DECL BOOL result
3 ...
4 result=StrToPOS("{X 10, Y 50.7, S 6, T 19}",value)
5 result=StrToPOS("{X10}",value)
6 result=StrToPOS("{E1 100}",value)
```

```
7 result=StrToPOS("{X 10, E1 100}",value)
8 result=StrToPOS("{E1 100, X 10}",value)
```

Line	Description
4	Conversion is possible. The return value *result* is TRUE. The value after conversion is "X 10, Y 50.7, S 6, T 19".
5	Conversion is not possible. The return value *result* is FALSE. It must be spelled "X 10", i.e. with a space.
6	Conversion is not possible, as E1 is not a component of POS. The return value *result* is FALSE.
7	Conversion is possible. The value of E1 is lost. The return value *result* is TRUE. The value after conversion is "X 10".
8	The same result as in line 7

11.17.1.6 StrToE3POS: converting character string to E3POS

Description

The function **StrToE3POS()** can be used to convert a character string to a variable of data type E3POS.

Syntax

result = `StrToE3POS(`*string*`,` *value*`)`

Explanation of the syntax

Element	Description
result	Type: BOOL Variable for the return value • TRUE: Conversion successful • FALSE: Conversion not successful
string	Type: CHAR Character string that is to be converted to a different data type. Maximum 256 characters. • If *string* represents an aggregate, the individual components must be separated by commas. • Missing components are not initialized and thus remain without a value. • If there are already components present in the target variable, they are deleted.
value	Type: E3POS Variable for the value after conversion

Examples

```
1 DECL E3POS value
2 DECL BOOL result
3 ...
```

```
4 result=StrToE3POS("{X 10, Y 50.7, S 6, E1 100}",value)
5 result=StrToE3POS("{X10}",value)
6 result=StrToE3POS("{E4 100}",value)
7 result=StrToE3POS("{X 10, E4 100}",value)
8 result=StrToE3POS("{E4 100, X 10}",value)
```

Line	Description
4	Conversion is possible. The return value *result* is TRUE. The value after conversion is "X 10, Y 50.7, S 6, E1 100".
5	Conversion is not possible. The return value *result* is FALSE. It must be spelled "X 10", i.e. with a space.
6	Conversion is not possible, as E4 is not a component of E3POS. The return value *result* is FALSE.
7	Conversion is possible. The value of E4 is lost. The return value *result* is TRUE. The value after conversion is "X 10".
8	The same result as in line 7

11.17.1.7 StrToE6POS: converting character string to E6POS

Description

The function **StrToE6POS()** can be used to convert a character string to a variable of data type E6POS.

Syntax

result = `StrToE6POS(`*string*`,` *value*`)`

Explanation of the syntax

Element	Description
result	Type: BOOL Variable for the return value • TRUE: conversion successful • FALSE: conversion not successful
string	Type: CHAR Character string that is to be converted to a different data type. Maximum 256 characters. • If *string* represents an aggregate, the individual components must be separated by commas. • Missing components are not initialized and thus remain without a value. • If there are already components present in the target variable, they are deleted.
value	Type: E6POS Variable for the value after conversion

Examples

```
1 DECL E6POS value
2 DECL BOOL result
3 ...
4 result=StrToE6POS("{X 10, Y 50.7, S 6, E4 100}",value)
5 result=StrToE6POS("{X10}",value)
6 result=StrToE6POS("{A1 90, E4 100}",value)
```

Line	Description
4	Conversion is possible. The return value *result* is TRUE. The value after conversion is "X 10, Y 50.7, S 6, E4 100".
5	Conversion is not possible. The return value *result* is FALSE. It must be spelled "X 10", i.e. with a space.
6	Conversion is not possible. Even though E4 is a component of E6POS, conversion is not possible due to the combination with A1 (or another value for rotational axes). The return value *result* is FALSE.

11.17.1.8 StrToBOOL: converting character string to BOOL

Description

The function **StrToBOOL()** can be used to convert a character string to a variable of data type BOOL.

Syntax

result = StrToBOOL(*string*, *value*)

Explanation of the syntax

Element	Description
result	Type: BOOL Variable for the return value • TRUE: conversion successful • FALSE: conversion not successful
string	Type: CHAR Character string that is to be converted to a different data type. Maximum 256 characters.
value	Type: BOOL Variable for the value after conversion

Examples

```
1 DECL BOOL result, value
2 ...
3 result=StrToBOOL("1", value)
4 result=StrToBOOL("TRUE", value)
```

Line	Description
3	Conversion is not possible, as the character string “1” does not match the data type BOOL. The return value *result* is FALSE.
4	Conversion is possible. The return value *result* is TRUE. The value after conversion is TRUE.

11.17.1.9 StrToINT: converting character string to INT

Description

The function **StrToINT()** can be used to convert a character string to a variable of data type INT.

Syntax

result = `StrToINT(`*string*`,` *value*`)`

Explanation of the syntax

Element	Description
result	Type: BOOL Variable for the return value • TRUE: conversion successful • FALSE: conversion not successful
string	Type: CHAR Character string that is to be converted to a different data type. Maximum 256 characters.
value	Type: INT Variable for the value after conversion

Examples

```
1 DECL INT value
2 DECL BOOL result
3 ...
4 result=StrToINT("1100", value)
5 result=StrToINT("1 100", value)
6 result=StrToINT("TRUE", value)
```

Line	Description
4	Conversion is possible. The return value *result* is TRUE. The value after conversion is 1100.
5	Conversion is not possible. The return value *result* is FALSE.
6	Conversion is not possible, as the character string “TRUE” does not match the data type INT. The return value *result* is FALSE.

11.17.1.10 StrToREAL: converting character string to REAL

Description

The function **StrToREAL()** can be used to convert a character string to a variable of data type REAL.

Syntax

result = StrToREAL(*string*, *value*)

Explanation of the syntax

Element	Description
result	Type: BOOL Variable for the return value • TRUE: conversion successful • FALSE: conversion not successful
string	Type: CHAR Character string that is to be converted to a different data type. Maximum 256 characters.
value	Type: REAL Variable for the value after conversion

Examples

```
1 DECL REAL value
2 DECL BOOL result
3 ...
4 result=StrToREAL("45.8", value)
5 result=StrToREAL("TRUE", value)
```

Line	Description
4	Conversion is possible. The return value *result* is TRUE.
5	Conversion is not possible, as the character string “TRUE” does not match the data type REAL. The return value *result* is FALSE.

11.17.1.11 StrToSTRING: converting nested character string to simple character string (CHAR)

Description

The function **StrToSTRING()** can be used to convert a nested character string to a simple character string.

Syntax

result = StrToSTRING(*string*, *value*)

Explanation of the syntax

Element	Description
result	Type: BOOL Variable for the return value • TRUE: conversion successful • FALSE: conversion not successful
string	Type: CHAR Nested character string that is to be converted to a simple character string. Maximum 256 characters.
value	Type: CHAR Variable for the value after conversion

Examples

```
1 DECL CHAR value[9]
2 DECL BOOL result
3 ...
4 result=StrToSTRING("wonderful", value[])
5 result=StrToSTRING(" '"'wonderful'"' ", value[])
```

Line	Description
4	Conversion is not possible because the output value is a simple value rather than a nested character string. The return value *result* is FALSE. The value of *value* is "".
5	Conversion is possible. The return value *result* is TRUE. • Prior to conversion: " "wonderful" " (character string in character string) • After conversion: "wonderful"

Here is another example to illustrate the character count and handling of the quotation marks:

```
1 CHAR v_before[13]
2 CHAR v_after[9]
3 BOOL result
4 v_before[]=" '"'wonderful'"' "
5 v_after[]=" "
6 ...
7 result=StrToSTRING(v_before[],v_after[])
```

Line	Description
1	The array index must be (at least) “13” because the 2 inner spaces and the 2 inner double quotation marks need to be added to the 9 letters. The single quotation marks (around the inner double quotation marks) are necessary, but do not count as characters.
7	Conversion is possible. The return value *result* is TRUE. • Prior to conversion: " "wonderful" " (character string in character string) • After conversion: "wonderful"

11.17.2 StrDeclLen: 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

11.17.3 StrLen: string variable length after initialization

Description

The function `StrLen()` determines the current length of the character string of a string variable.

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

11.17.4 StrClear: 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. The return value is always TRUE.
StrVar[]	Type: CHAR array Variable whose character string is to be deleted

Example

```
1 CHAR my_char[7]
2 BOOL result
3 my_char[]="example"
4 ...
5 result=StrClear(my_char[])
```

Line	Description
3	The value of *my_char* is "example".
5	The value of *my_char* is now "".

11.17.5 StrAdd: 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

11.17.6 StrFind: 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.

Element	Description
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

11.17.7 StrComp: 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[]

Element	Description
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

11.17.8 StrCopy: 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. The return value is always TRUE.
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 the assignment. It is necessary to enable the controller-internal submit tasks to be executed. If the 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.
The reason is: 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.

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. It has been stopped.
RESET	A program is selected in the submit interpreter. It has been stopped and reset.

State	Description
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 ACTIVE. All others are in the FREE state.
	**Green with exclamation mark** At least one submit interpreter is ACTIVE. At least one is in the RESET or STOP state.
	**Red** At least one submit interpreter is in the STOP state. All others are in the FREE state.
	**Red with exclamation mark** At least one submit interpreter is in the STOP state. At least one is in the RESET state. No submit interpreter is ACTIVE.
	**Yellow** At least one submit interpreter is in the RESET state. All others are in the FREE 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.

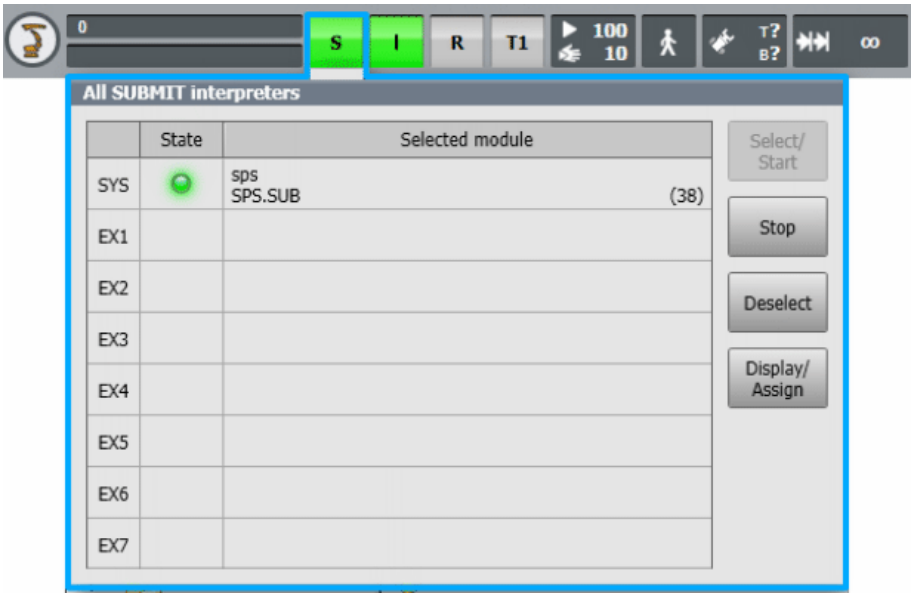


Fig. 12-2: Overview of All SUBMIT interpreters

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 698*)
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 697*)
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 696*)

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 697*)
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

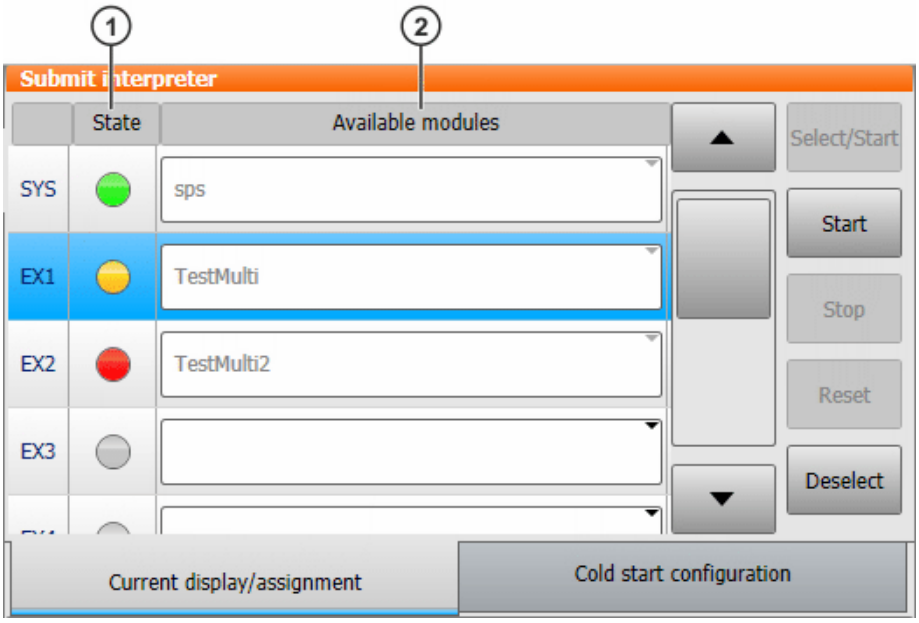


Fig. 12-3: Current display/assignment

Item	Description	
1	State of the interpreter (>>> *"States" Page 691*)	
		**Green**: ACTIVE
		**Yellow**: RESET
		**Red**: STOP, i.e. stopped by the user
		**Gray**: FREE
		**With red dot**: A problem occurred on starting the submit.

Item	Description
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 343*) 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 the assignment. It is necessary to enable the controller-internal submit tasks to be executed. If the 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.

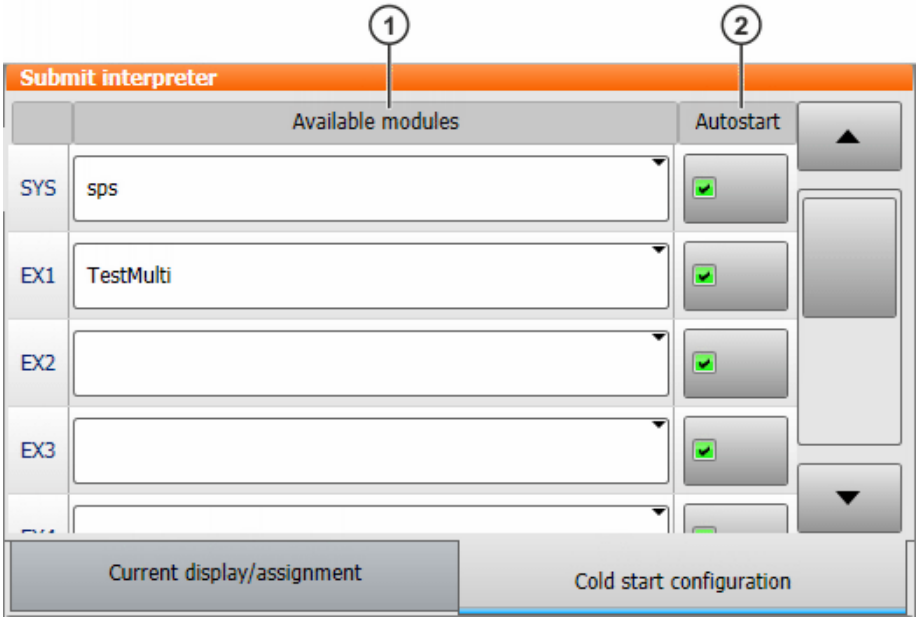


Fig. 12-4: Cold start configuration

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 343*) 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.

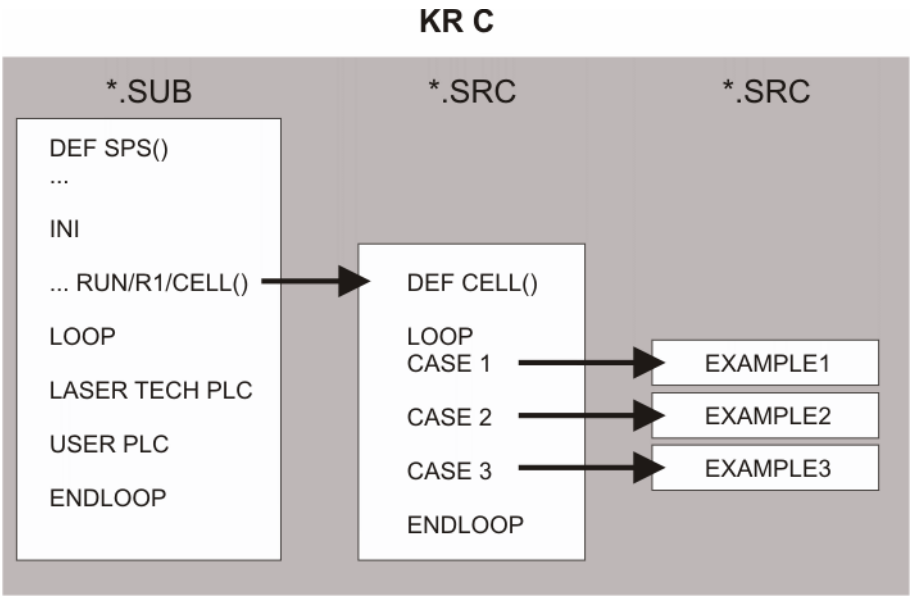


Fig. 12-5: sps.sub selects CELL.SRC in the robot interpreter

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 705*)
- $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 692*)

  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.

  (>>> *11.8.8.3 "$ERR" Page 601*)

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 705*)

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 Selection of 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

The variable $PROG_INFO[] groups together the states of all interpreters. Each array element has the STRUCTURE type PROG_INFO and refers to an interpreter.

Writability

The system variable is write-protected.

Syntax

proginfo `= $PROG_INFO[`*index*`]`

Explanation of the syntax

Element	Description
index	Type: INT Index of the interpreter • 1: Robot interpreter • 2: System submit interpreter • 3: Extended submit interpreter 1 • 4: Extended submit interpreter 2 • Etc.
proginfo	Type: STRUC PROG_INFO State of the relevant 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[]	Type: CHAR[] Name of the selected program
P_STATE	Type: ENUM PRO_STATE Program status • #P_ACTIVE: Program is selected and is running. • #P_FREE: Program is deselected. • #P_END: Program is selected and the end of the program has been reached. • #P_RESET: Program is selected and has been stopped and reset. • #P_STOP: Program is selected and has been stopped.

Element	Description
P_MODE	Type: ENUM PRO_MODE Program run mode • #GO: The program is executed through to the end without stopping. • #MSTEP: Motion Step The program is executed with a stop at each motion point, including auxiliary points and the points of a spline segment. The program is executed without advance processing. • #ISTEP: Incremental Step 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 program is executed without advance processing. • #BSTEP: Backward motion This program run mode is automatically selected if the Start backwards key is pressed. It is not possible to select a different mode. The response is like that for #MSTEP. • #PSTEP: Program Step The program is executed step by step without advance processing. Subprograms are executed completely. • #CSTEP: Continuous Step Approximate positioning points are executed with advance processing and approximated. Exact positioning points are executed without advance processing and with a stop after the motion instruction.
PRO_IP_NAME[]	Type: CHAR[] Name of the current module
PRO_IP_SNR	Type: INT Block number in which the interpreter is currently located

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 713*)
- Filter (>>> *13.1.3 "Filter tab" Page 715*)

13.1.2 “Log” tab

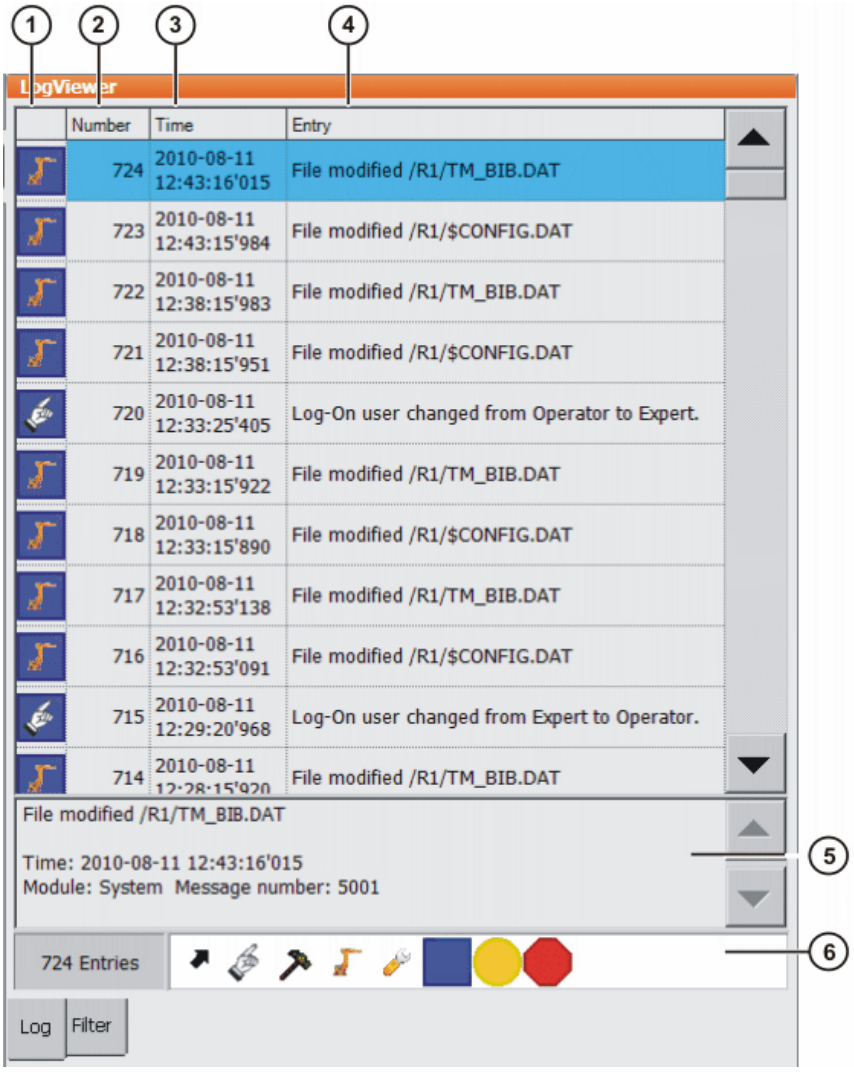


Fig. 13-1: Logbook, Log tab

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 715*) Required user rights: Function group **Archive to local HDD/SSD**
Refresh	Refreshes the log display.

13.1.3 Filter tab

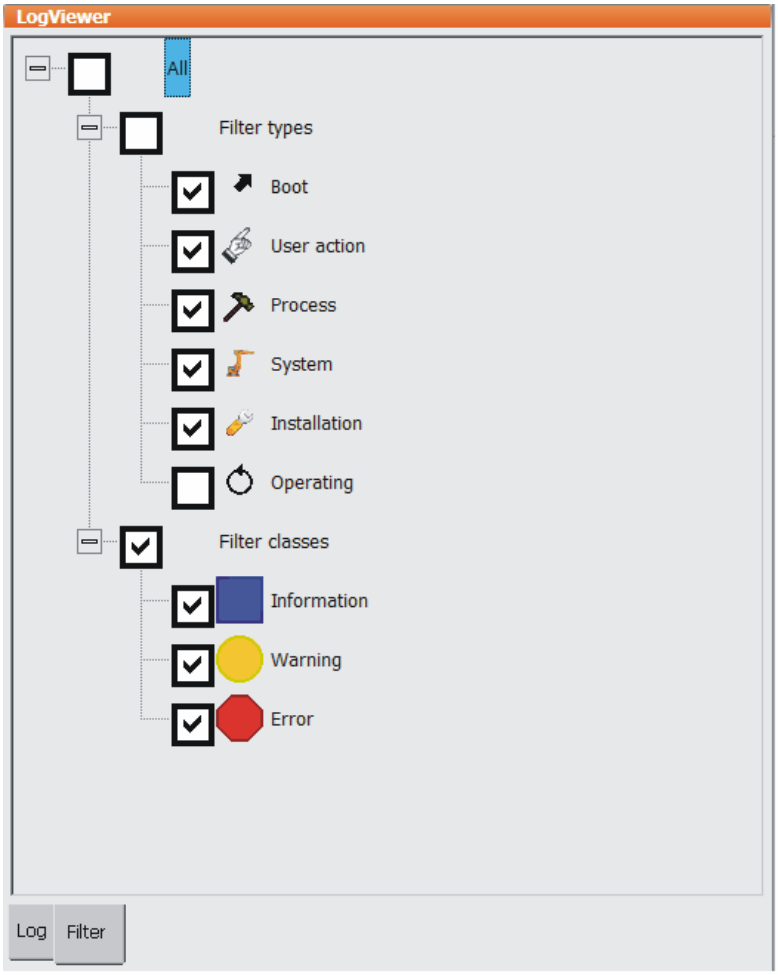


Fig. 13-2: Logbook, Filter tab

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

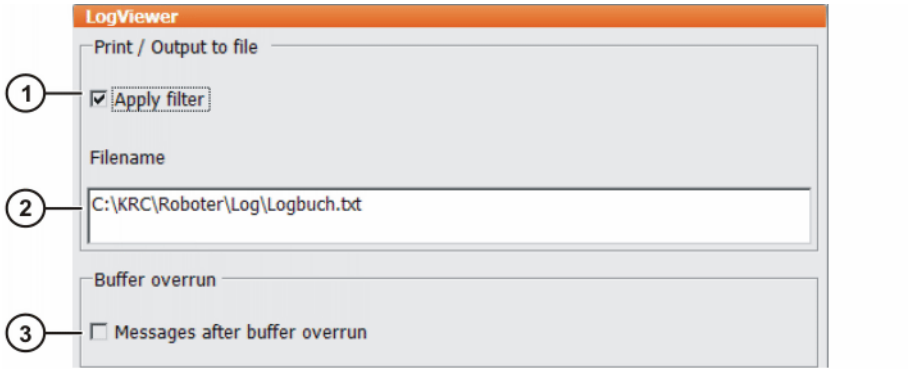


Fig. 13-3: Logbook configuration window

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 Display the moment load of the robot

Description

The robot controller calculates an indicator value for the moment load of the overall mechatronic robot system based on the load of individual components (including motors, gear units). The value also includes factors that cannot be predicted during the check with KUKA.Load, e.g. the actual load distribution during load sharing.

The indicator value is polled every interpolation cycle (= 12 ms). The **Instantaneous load** window displays the value relative to a configurable time interval.

NOTICE

Damage to property due to overloading
If the **Instantaneous load** window indicates an overload, damage to property may result. The longer the robot is in the red zone overall, and the higher the overload value, the more likely damage is to be expected.

- Recommendation: Reduce load
  - e.g. check and optimize programming, especially approximate positioning and accelerations
  - e.g. check and optimize load data
- If the robot does not reach the green zone despite the reduced load, the user has the option of contacting KUKA Customer Support about this.

Precondition

Only necessary if the display is to be adapted:

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

Procedure

- Main menu > **Diagnosis** > **Instantaneous load**
  The **Instantaneous load** window opens.

“Instantaneous load” window

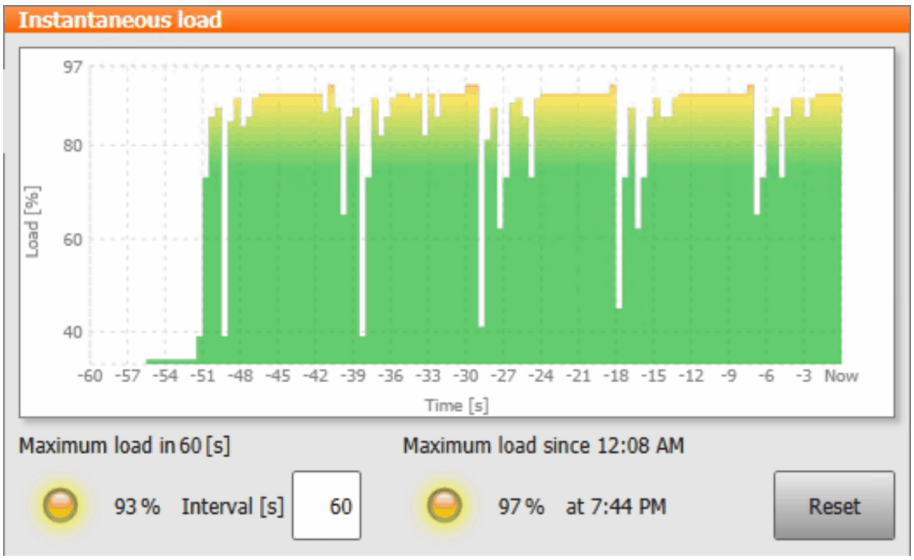


Fig. 13-4: “Instantaneous load” window

The display supplies a value regardless of the status of the drives.

Element	Description
Maximum load in [Interval [s]]	Maximum load that occurred in the last interval (sliding maximum) The sliding maximum is reset by the following actions: Restart with hibernate, reconfiguration of the I/O drivers, modification of the machine data
Interval [s]	The interval length can be modified. • **10 … 300 s** Default: 60 s
Maximum load since [time]	Maximum load since the last system start or since actuation of the **Reset** button (long-term maximum)
Reset	Resets the long-term maximum.
LEDs	• **Green**: Load < 95% • **Yellow**: Load 95 ... ≤ 100% Load bordering on overload. • **Red**: Load > 100% Overload

13.3 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

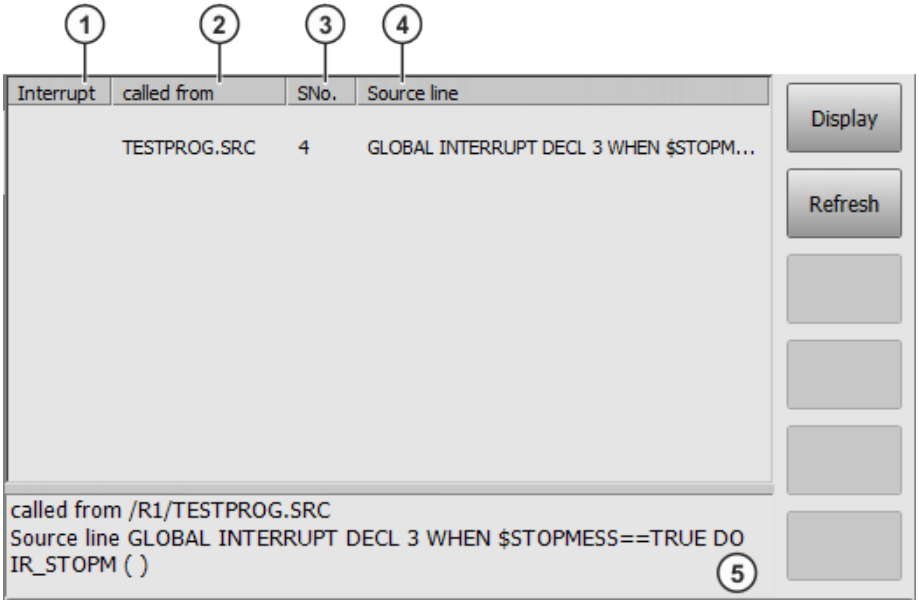


Fig. 13-5: “Caller stack” window

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

13.4 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

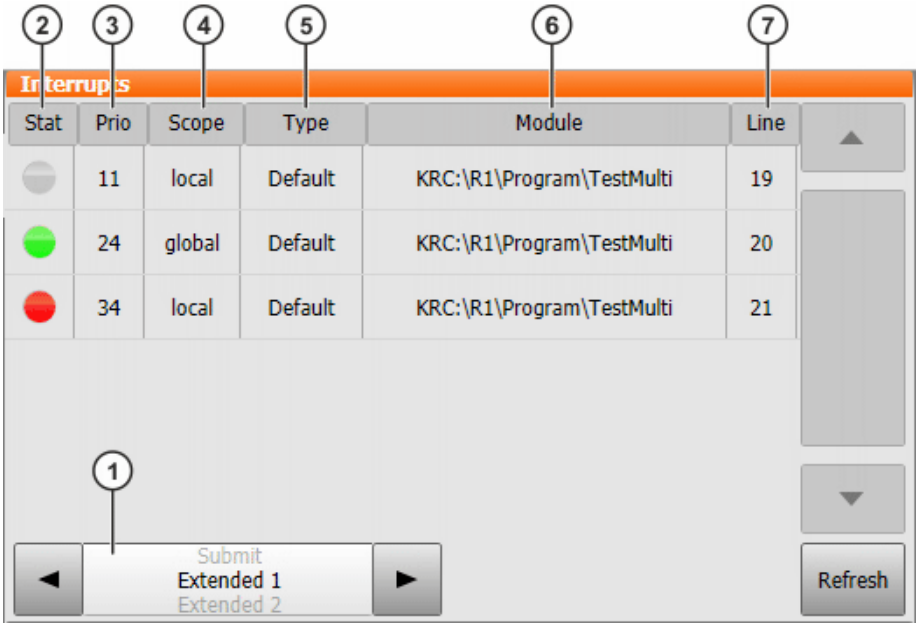


Fig. 13-6: Interrupts

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.

13.5 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.

13.6 Automatically compressing data for error analysis (diagnosis package “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.

(>>> *7.9 "Archiving and restoring data" Page 357*)

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 751*)

14.1 System requirements

The System Software 8.7 can be run on the following robot controller:

- KR C5
- Windows 10 IoT, V5.1.x or higher
- With “Performance” system board

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 8.7.0 to KSS 8.7.1.

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 8.6 to KSS 8.7. 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 725*)
- From the network
  (>>> *14.3.2 "Update from the network" Page 726*)

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 Set-up.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.

15 Appendix

15.1 Checklists for safety acceptance

The checklists here serve merely as examples of checklists for safety acceptance. It is permissible to carry out and document safety acceptance using user-specific checklists.

15.1.1 Precondition for safety acceptance based on the checklists

- Mechanical and electrical installation of the industrial robot have been completed.
- Safety configuration is completed.
- The safety maintenance technician is trained. (Training course at KUKA College)

The system integrator is responsible for the design of the cell.
The safety maintenance technician uses the values and configurations supplied by the system integrator to configure the robot and tests whether the safety functions work as specified. The safety maintenance technician does not perform a safety assessment of the system.

15.1.2 Checklist: Robot and system

The inspection items of this checklist must be completed and confirmed in writing by the system integrator.

Checklist

- Serial number of the robot: _____
- Serial number of the robot controller: _____
- Name of system integrator: _____

No.	Inspection item	OK
1	The industrial robot is in flawless mechanical condition and correctly installed and fastened in accordance with the assembly or operating instructions of the robot.	
2	The permissible rated payload of the robot has not been exceeded.	
3	There are no foreign bodies or defective or loose parts on the industrial robot.	
4	All safety equipment required for the system and robot is correctly installed and operational.	
5	The power supply ratings of the industrial robot correspond to the local supply voltage and mains type, and the machine data correspond to these.	
6	The connecting cables are correctly connected and the connectors are locked.	
7	The ground conductor and the equipotential bonding cable are sufficiently rated and correctly connected.	
8	The system meets all the relevant laws, regulations and norms valid for the installation site.	

Remarks / deviations

By signing, the signatory confirms the correct and complete performance of the safety acceptance test.

Place, date	Signature
Place, date	Signature
Place, date	Signature

15.1.3 Checklist: Safety functions

The inspection items of this checklist must be completed and confirmed in writing by the safety maintenance technician.

Checklist

- Serial number of the robot: _____
- Serial number of the robot controller: _____
- Safety configuration:
  - Activation code: _____
  - Last changed (date, time): _____
  - Version: _____
- Name of safety maintenance technician: _____

No.	Inspection item	OK	Not relevant
1	Robot is mastered.		
2	If simulated axes are used: The safe Cartesian monitoring has been activated or deactivated according to the requirements and, if applicable, in accordance with a hazard assessment. (>>> *5.12 "Switching safe Cartesian monitoring on/off" Page 157*)		
3	The machine data have been checked and are appropriate for the robot used. (>>> *3.12.4.1 "Checking machine data and safety configuration" Page 51*) The machine data loaded must match the machine data on the identification plate of the robot.		

No.	Inspection item	OK	Not relevant
4	The machine data of the external axes have been correctly entered and checked. Checking instructions: • Move each external axis a defined distance by means of a PTP_REL motion, e.g. 90°. Carry out a visual inspection and check whether this distance is covered. • In the case of a KL, move the external axis a defined distance by means of a PTP_REL motion, e.g. 500 mm. Carry out a visual inspection and additionally monitor the display of the Cartesian actual position to check whether this distance is covered.		
5	The local and external safety functions have been checked and are functioning correctly. (>>> *3.12.4 "Start-up and recommissioning: KSS/VSS" Page 49*)		
6	Was the brake test successful? **Note:** These checks only need to be made if the brake test is active. The checks only need to be made for the axes that need to be tested by means of a brake test in accordance with a hazard assessment.		
	Axis A1		
	Axis A2		
	Axis A3		
	Axis A4		
	Axis A5		
	Axis A6		
	External axes		
7	Operator safety acknowledgement has been checked and is functioning correctly. (>>> *3.6.3 ""Operator safety" signal: KSS" Page 36*) (>>> *3.6.4 ""Operator safety" signal: VSS" Page 36*)		
8	Peripheral contactor (US2) has been checked and switches at the right time. **Note:** Further information about checking the peripheral contactor is contained in the assembly instructions of the robot controller.		
9	Have the maximum axis velocities for T1 been checked? The associated checklist must be completed fully and confirmed in writing. (>>> *15.1.4 "Checklist: Maximum axis velocities for T1" Page 732*)		
10	The values for the parameters **Braking time** and **Position tolerance** have been checked in the diagnostic monitor and correctly saved. (>>> *6.5 "Checking the values for the safe axis monitoring functions" Page 244*)		
11	If an existing safety configuration has been changed: A change log has been created and checked.		

Remarks / deviations

☐ : Must be checked by system integrator

By signing, the signatory confirms the correct and complete performance of the safety acceptance test.

Place, date	Signature
Place, date	Signature
Place, date	Signature

15.1.4 Checklist: Maximum axis velocities for T1

The inspection items of this checklist must be completed and confirmed in writing by the safety maintenance technician.

The trace function (oscilloscope) can be used to determine the axis velocities.

Precondition

- Override reduction for velocity limits is deactivated:
  - $SR_VEL_RED = FALSE

Checklist

- Serial number of the robot: _____
- Serial number of the robot controller: _____
- Safety configuration:
  - Activation code: _____
  - Last changed (date, time): _____
  - Version: _____
- Name of safety maintenance technician: _____

Specified value:

- Value specified by cell planner, design engineer

Configured value:

- Value entered in the safety configuration

Test value:

- Value with which the test was carried out

No.	Axis name	Inspection item	OK	Not relevant
The maximum axis velocity for T1 has been correctly configured and checked for each axis.				
1		Specified value: _____ °/s or mm/s Configured value: _____ °/s or mm/s Test value: _____ °/s or mm/s		
2		Specified value: _____ °/s or mm/s Configured value: _____ °/s or mm/s Test value: _____ °/s or mm/s		
3		Specified value: _____ °/s or mm/s Configured value: _____ °/s or mm/s Test value: _____ °/s or mm/s		
4		Specified value: _____ °/s or mm/s Configured value: _____ °/s or mm/s Test value: _____ °/s or mm/s		
5		Specified value: _____ °/s or mm/s Configured value: _____ °/s or mm/s Test value: _____ °/s or mm/s		
6		Specified value: _____ °/s or mm/s Configured value: _____ °/s or mm/s Test value: _____ °/s or mm/s		
7		Specified value: _____ °/s or mm/s Configured value: _____ °/s or mm/s Test value: _____ °/s or mm/s		
8		Specified value: _____ °/s or mm/s Configured value: _____ °/s or mm/s Test value: _____ °/s or mm/s		
9		Specified value: _____ °/s or mm/s Configured value: _____ °/s or mm/s Test value: _____ °/s or mm/s		
10		Specified value: _____ °/s or mm/s Configured value: _____ °/s or mm/s Test value: _____ °/s or mm/s		
11		Specified value: _____ °/s or mm/s Configured value: _____ °/s or mm/s Test value: _____ °/s or mm/s		
12		Specified value: _____ °/s or mm/s Configured value: _____ °/s or mm/s Test value: _____ °/s or mm/s		

No.	Axis name	Inspection item	OK	Not relevant
13		Specified value: _____ °/s or mm/s Configured value: _____ °/s or mm/s Test value: _____ °/s or mm/s		
14		Specified value: _____ °/s or mm/s Configured value: _____ °/s or mm/s Test value: _____ °/s or mm/s		
15		Specified value: _____ °/s or mm/s Configured value: _____ °/s or mm/s Test value: _____ °/s or mm/s		
16		Specified value: _____ °/s or mm/s Configured value: _____ °/s or mm/s Test value: _____ °/s or mm/s		

Remarks / deviations

☐ : Must be checked by system integrator

By signing, the signatory confirms the correct and complete performance of the safety acceptance test.

_____	_____
Place, date	Signature
_____	_____
Place, date	Signature
_____	_____
Place, date	Signature

15.2 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.

15.2.1 Menu item File

Menu sequence	Function group
File > **Archive** > **USB (KCP)** > **All**	**Archive to USB drives**
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**

Menu sequence	Function group
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**

15.2.2 Menu item Configuration

Menu sequence	Function group
Configuration > Inputs/outputs > Automatic External Subfunctions: (>>> *15.2.2.1 "Automatic External" Page 737*)	None
Configuration > Inputs/outputs > I/O drivers Subfunctions: (>>> *15.2.2.2 "I/O drivers" Page 737*)	None
Configuration > SUBMIT interpreter	**General configuration**
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: (>>> *15.2.2.3 "Cartesian workspaces/Axis-specific workspaces" Page 737*)	None
Configuration > Miscellaneous > Event planner Subfunctions: (>>> *15.2.2.4 "Event planner" Page 737*)	**General configuration**
Configuration > Miscellaneous > Point coordinate correction limit Subfunctions: (>>> *15.2.2.5 "Point coordinate correction limit" Page 738*)	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

Menu sequence	Function group
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

15.2.2.1 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

15.2.2.2 I/O drivers

Operator control element	Function group
Reconfigure	**General configuration**

15.2.2.3 Cartesian workspaces/Axis-specific workspaces

Operator control element	Function group
Toggles between the pages via: **Signal/Cartesian/Axis-specific**	None
Save	**General configuration**

15.2.2.4 Event planner

Operator control element	Function group
Refresh	None
Save	**General configuration**

15.2.2.5 Point coordinate correction limit

Operator control element	Function group
Correction limit active	**General configuration**
Cartesian distance	
Angle of rotation	

15.2.3 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: (>>> *15.2.2.1 "Automatic External" Page 737*)	None
Display > **Inputs/outputs** > **I/O drivers** Subfunctions: (>>> *15.2.2.2 "I/O drivers" Page 737*)	None
Display > **Actual position**	None
Display > **Variable** > **Single** Subfunctions: (>>> *15.2.3.1 "Variable display" Page 738*)	None
Display > **Variable** > **Overview** > **Display** Subfunctions: (>>> *15.2.3.2 "Variable overview" Page 739*)	None
Display > **Variable** > **Overview** > **Configuration** Subfunctions: (>>> *15.2.3.2 "Variable overview" Page 739*)	**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

15.2.3.1 Variable display

Operator control element	Function group
Refresh	None
Set value	**General configuration**

15.2.3.2 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

15.2.4 Menu item Diagnosis

Menu sequence	Function group
Diagnosis > **Trace** > **Configuration** Subfunctions: (>>> *15.2.4.1 "Trace" Page 739*)	**Diagnostic functions**
Diagnosis > **Trace** > **Oscilloscope**	**Diagnostic functions**
Diagnosis > **Logbook** > **Display** Subfunctions: (>>> *15.2.4.2 "TraceLogbook" Page 739*)	**Diagnostic functions**
Diagnosis > **Logbook** > **Configuration**	**General configuration**
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**

15.2.4.1 Trace

Operator control element	Function group
Start trace	**Diagnostic functions**
Stop Trace	**Diagnostic functions**
Trigger	**Diagnostic functions**
All other operator control elements	**General configuration**

15.2.4.2 TraceLogbook

Operator control element	Function group
Export	**Archive to local HDD/SSD**
Refresh	None
OK	**General configuration**

Operator control element	Function group
Save	**General configuration**

15.2.5 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**
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: (>>> *15.2.5.1 "Robot data" Page 741*)	None

Menu sequence	Function group
Start-up > **Network configuration** Subfunctions: (>>> *15.2.5.2 "Network configuration" Page 741*)	None
Start-up > **Additional software** Subfunctions: (>>> *15.2.5.3 "Additional software" Page 741*)	**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

15.2.5.1 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	

15.2.5.2 Network configuration

Element	Function group
“**Internal subnets**” tab	**Critical configurations**
[All elements outside of the tab **Internal subnets**]	**General configuration**

15.2.5.3 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**

Operator control element	Function group
Path selection	**General configuration**
Save	**General configuration**
Cancel	None

15.2.6 Menu item Shutdown

Menu item	Function group
Shutdown	None

Subfunctions:

Element	Function group
Start type	**Critical configurations**
Power-fail wait time [s]	**Critical configurations**
Force cold start	**General configuration**
Reload files	**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**

15.2.7 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

15.2.8 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	

Button	Function group
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**	
Archive all > **USB (KCP)**	**Archive to USB drives**
Archive all > **USB (cabinet)**	
Archive all > **Network**	**Archive to network**
EDITOR	None

15.2.9 Navigator: Edit menu

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**
Archive > **USB (cabinet)**	& **Partial archiving**
Archive > **Network**	**Archive to network** & **Partial archiving**
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**

15.2.10 Editor: button bar

Button	Function group
Change	**General KRL program changes**
Commands Subfunctions:	**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**
Edit Subfunctions:	None
Last command	**General KRL program changes**

15.2.11 “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

15.2.12 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	

Element	Function group
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

15.2.13 Wait messages

Button	Function group
Simulate	Block selection

15.2.14 Editor: Edit menu

Menu item	Function group
FOLD > Open/close current FOLD	None
FOLD > Open all FOLDs	
FOLD > 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

Menu item	Function group
Delete	**General KRL program changes**
Search	None
Replace	**Critical KRL program changes**
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 > Axis mirroring	
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

15.2.15 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 > HOP	**New motion range inline forms**
Motion parameters > Collision detection	None
Motion parameters > Spline trigger	**New motion range inline forms**
Motion parameters > Spline Stop Condition	
Logic > WAIT	None
Logic > WAITFOR	
Logic > OUT > OUT	
Logic > OUT > PULSE	
Logic > OUT > SYN OUT	
Logic > OUT > SYN PULSE	

Button	Function group
Logic > Analog output > Static	None
Logic > Analog output > Dynamic	
Logic > TIMER	
Logic > Variable declarations	
Logic > Interrupt declaration	
Logic > Switch the interrupt	
Logic > IF ... THEN	
Logic > WHILE ... ENDWHILE	
Logic > REPEAT ... UNTIL	
Logic > LOOP ... ENDLOOP	
Logic > FOR ... ENDFOR	
Logic > SWITCH ... CASE	
Comment > Normal	General KRL program changes
Comment > Stamp	
Trace > Activate trace	Diagnostic functions
Trace > Deactivate trace	

15.2.16 smartHMI: status bar

Overview

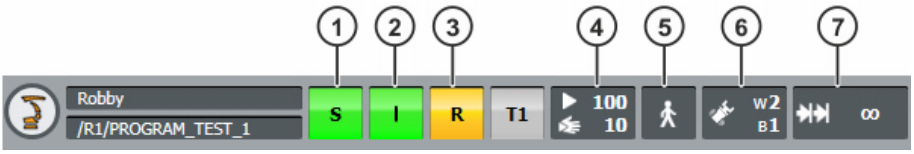


Fig. 15-1: Indicators in the status bar relating to function groups

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**

15.2.17 smartHMI: left sidebar

Element	Function group
WorkVisual icon > **Open** button Subfunctions: (>>> *15.2.11 ""Project management" window" Page 744*)	None

15.2.18 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**

Element	Function group
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
- The greatest possible amount of information 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
  – System software diagnosis package

    Additionally for KUKA Sunrise: Existing projects including applications

    For versions of KUKA System Software older than V8: Archive of the software (Diagnosis package is not yet available here.)
  – Application used
  – External axes used

16.2 KUKA Customer Support

The contact details of the local subsidiaries can be found at:
www.kuka.com/customer-service-contacts

Index

#BSTEP....378
#CSTEP....378
#GO....377
#IGNORE....436, 437
#ISTEP....377
#MSTEP....377
#PSTEP....378
$....546
$ACCU_STATE....134
$ADAP_ACC....393
$ADVANCE....378
$ALARM_STOP....287
$ALARM_STOP_INTERN....290
$ANIN....527
$ANOUT....527
$AXIS_HOME....293
$BRAKE_SIG....311
$BRAKES_OK....326
$BRAKETEST_MONTIME....325
$BRAKETEST_REQ_EX....325
$BRAKETEST_REQ_INT....326
$BRAKETEST_WARN....326
$BRAKETEST_WORK....326
$BRK_DEL....670
$BWD_INFO....391
$BWDSTART....391
$CHCK_MOVENA....284
$CIRC_MODE....439
$CIRC_TYPE....438
$CMD....707
$COLL_ALARM....392, 393
$COLLMON_STARTUP_MAX....412
$CONF_MESS....284
$CONST_VEL....585
$CONST_VEL_C....585
$COOLDOWN_TIME....271
$COULD_START_MOTION....68
$CRIT_PERI_ACK....153
$CRIT_PERI_ACK_REQ....154
$DIST_LAST....647
$DIST_NEXT....646
$DRIVES_OFF....285
$DRIVES_ON....285
$ECO_LEVEL....256
$ENERGY_CONFIG_STATE....127
$ENERGY_INTERIM....129
$ENERGY_MEASURING....125
$ENERGY_PERIOD....128
$ENERGY_TOTAL....129
$ENERGYMODULE....126
$ERR....601, 704
$EX_AX_IGNORE....589
$EXT_START....283
$H_POS....292
$HOLDING_TORQUE....312, 313
$I_O_ACT....285
$I_O_ACTCONF....289
$IMPROVED_COLLMON....393
$IN_HOME....292, 293
$INTERPRETER....704, 705
$LDC_CONFIG....220
$LDC_LOADED....219
$LDC_RESULT....220
$LOAD_BWINI....391
$MOVE_ENABLE....283, 284
$NEAR_POSRET....293
$ON_PATH....293
$ORI_TYPE....418, 436
$PAL_MODE....160
$PATHTIME....581
$PERI_RDY....68, 288
$POS_ACT....650
$PRO_ACT....290
$PRO_IP....717
$PRO_IP....704
$PRO_MODE....378
$PRO_MODE....704
$PRO_MOVE....292
$PRO_NAME....704
$PRO_STATE0....704
$PROG_INFO....705
$RC_RDY1....287
$ROB_CAL....288
$ROB_STOPPED....294
$ROBRUNTIME....120, 122
$SPL_ORI_JOINT_AUTO....438
$STOP_CONST_VEL_RED....584
$STOPMESS....289
$TIMER_FLAG....529
$TOOL_DIRECTION....189
$TOOL_DIRECTION_LIN_CIRC....189
$TORQUE_AXIS_ACT....310, 313
$TORQUE_AXIS_LIMITS....311
$TORQUE_AXIS_MAX....311
$TORQUE_AXIS_MAX_0....311
$US2_VOLTAGE_ON....151
$USER_SAF....70, 288
$VEL_APPL....591
$VW_BACKWARD....391
$VW_CYCFLAG....391
$VW_MOVEMENT....391
$WAIT_FOR_ON0....704
$WAIT_FOR0....704
$WARMUP_CURR_LIMIT....271
$WARMUP_MIN_FAC....271
$WARMUP_RED_FAC_ACT....271
$WARMUP_RED_VEL....271
$WARMUP_SLEW_RATE....271
$WARMUP_TIME....271
_TYP....553
_TYPE....554
“Drives ready” interface....45
”Drives ready” interface....43
”Drives ready” lamp....43
3-point (method)....206
6D mouse....62

A

A6, mastering position....179
ABC 2-point (method)....205
ABC world (method)....204
Absolute value (mathematical function)....658
Accessories....23, 27
Activating, project....361
Activation code, safety configurationSafety configuration, activation code....246
ACTIVE (state)....691
Actual position....106
Addition....649
Addition, geometric....650
Administrator (user group)....83
Advance run....378
Advance run stop....417, 594
Analog inputs....611
Analog outputs....612
ANIN OFF....611
ANIN ON....611
ANOUT OFF....612
ANOUT ON....612
ANSI/UL 1740....43
Appendix....729
APPL_RUN....291
Approximate positioning....417, 456
Approximate positioning, homogeneous....642
Approximate positioning, mixed....642
Arc cosine (mathematical function)....658
Arc sine....659
Arc tangent (mathematical function)....658
Archiving overview....357
Archiving, logbook....360
Archiving, network....359
Archiving, to USB stick....358
Areas of validity....549
ASYPTP....702
AUT (operating mode)....28
AUT EXT (operating mode)....28
Automatic (operating mode)
AUT....28
AUT EXT....28
Automatic External (operating mode)....45
Automatic mode....56
Auxiliary point....416, 557, 561
Axis limitation, mechanical....41
Axis monitoring functions, checking....244
Axis monitoring functions, configuring....238
Axis range....28
Axis, active....320
Axis, requested....320

B

Backup configuration (window)....371
Backup Manager....366
Backup Manager, configuring....371
Backup, option packages....366, 367
Backup, projects....366, 367
Backup, RDC data....366, 367
Backward motion (using “Start backwards” key)....386
Backward motion (using jog keys)....101
Backward motion, configuring....275
Backward, preventing start....660
BACKWARD_STEP....275
BASE....192
Base calibration....192
BASE coordinate system....85, 192
Battery state....134
BCO run....383
Bit operators....655
Block coincidence....383
Block pointer....346, 379
Block selection....384, 424
BRAKE....626, 631, 632
Brake check, automatic....320, 335
Brake closing time....669
Brake defect....47
BRAKE F....632
BRAKE FF....632
Brake release device....42
Brake test....319
Brake test, cycle time....320
Brake test, manual....332
Brake test, operating modes....319
Brake test, programs....321
Brake test, signals....324, 327
Brake test, teaching positions....328
Brake, defective....330, 332–334
BrakeTestAxes.src....321
BrakeTestBack.src....322, 329
BrakeTestPark.src....321, 329
BrakeTestReq.src....321
BrakeTestStart.src....322, 328
Braking distance....28
Branch, conditional....534, 597
BSTEP....378
Bus Power Off mode....75, 76
Bypassing workspace monitoring....105

C

C_APX....458, 562
Calibration....189
Calibration tolerances, defining....272
Calibration, base....192
Calibration, linear unit....212
Call by Reference....621
Call by Value....621
Caller stack....717
Caller stack (menu item)....718
CANCEL (command)....708
Cancel, program....345
CAST_FROM....648
CAST_TO....648
CCLOSE....648
CE mark....28
CELL.SRC....385
CHANNEL....648

Checklists for safety acceptance....729
CIOCTL....649
CIRC....557
CIRC motion....454
CIRC, motion type....416
CIRC_REL....561
Circular angle....443
Circular motion....557, 561
CK, monitoring....40
Cleaning work....57
Close all FOLDs (menu item)....351
Cold start....76, 693, 698
Cold start, initial....72, 75, 76, 725
CollDetect_UserAction....393
Collision detection....392, 402, 403
Collision detection, configuring....396
Command interface....707
Comment....353, 354
Commenting out....354
Communication....703
Comparison, data from kernel system and hard drive....319
Conditional branch....534, 597
Configuration....229
Configuration (menu item)....249
Configuring CELL.SRC....277
Connecting cables....23, 27
Connection manager....62
CONST....551
CONST_VEL....582
Constant velocity range....498, 582
Constants....550, 551
CONTINUE....594
Continuous Path....415
Controller name, display....68
Coordinate systems....85
Coordinate systems, angles....86
Coordinate systems, orientation....86
COPEN....649
Copy....356
Cosine (mathematical function)....658
Counterbalancing system....57
Counters, displaying....118
CP motions....415
CP spline block....468, 469, 565
CREAD....649
Creating a new folder....339
Creating a new program....339
Cut....356
CWRITE....649, 707

D

Danger zone....29
DAT....546
Data list....546
Data type, user-defined....550, 553, 554
Data types....547
DECL....551
Declaration of conformity....27
Declaration of incorporation....27, 28
Decommissioning....58
DEF line (menu item)....349
DEF line, displaying/hiding....349
DEFAULT....541, 608
DEFFCT ... ENDFCT....619
DELETE_BACKWARD_BUFFER....660
Deselecting, submit interpreters....694, 695
Detail view (ASCII) (menu item)....350
Detail view, activating....349
Diagnosis....713
Diagnosis package....358, 720, 751
Diagnostic monitor (menu item)....720
Dial gauge....175
Directory structure....340
Disclaimer....27
Display (menu item)....114
Displaying a variable, single....112, 114
Displaying the logbook....713
Displaying variables, in overview....114
Displaying, robot controller information....120
Displaying, robot information....120
Disposal....58
DISTANCE....634
Division....649
Documentation, industrial robot....21
Drives, switching on/off....69

E

EC declaration of conformity....27
Edit (button)....66
Editor....345
ELSE....534, 597
EMC Directive....28
EMERGENCY STOP....62
  External....29, 37
  Local....29, 37
EMERGENCY STOP device....37, 44, 46
  External....29, 37, 51
  Local....29, 37, 51
Enabling device....38, 44, 46
Enabling device, external....40
Enabling switches....38
ENDFCT....619
Endless loop....538, 598
ENDSPLINE....565, 566
Energy consumption, measuring....123
ENUM....553
Enumeration type....553
ERR_RAISE....604
EtherNet/IP interface....231
Even parity....286
EXIT....538, 594, 598
EXP_SCLOG.xml....247
EXP_SCTLCONFIG.xml....247
EXP_SCTLCRC.xml....246, 247
Expert (user group)....83
Expert Submit (template)....700
Export (button)....246

EXT (operating mode)....45
Extended submit....691
Extended submit interpreter....691
External axes....27, 120
External axis....31

F

FALSE....618
Faults....48
Field bus interface....231
Field bus, Ethernet-based....229
File list....340
File, properties....342
Filter....341
Find....356
Firmware, updating....131
First mastering....169, 180
Flags, displaying....116, 117
FLANGE coordinate system....86, 190
Folder, creating....339
Folder, properties....342
Folds....350
Folds, creating....352
Folds, displaying....350
Fonts....545
FOR....609
FOR ... TO ... ENDFOR....595
FOR ... TO ... ENDFOR (inline form)....540
FORWARD....661
Frame operation....650
FREE (state)....692
FREE_PRESELECTED (state)....692
FREE_UNSELECTED (state)....692
Function groups....251, 253
Function test....50
Function, calling....619
Function, syntax....619

G

General safety measures....46
Geometric addition....650
GET_AXESMASK....336
GET_BRAKETEST_TIME....337
GET_SIG_INF....662
Global....549
GLOBAL (interrupt declaration)....626
Global point....510
Global_Points.dat....510
GO (program run mode)....377
GOTO....596

H

HALT....597
Hard disk space....121
Hardware, options....149
Hardware, updating firmware....131
Hazardous substances....58
Header....340
Help, messages....78
Hibernate....76
HOME position....348
Homogeneous approximate positioning....642
HOP motion....460

I

I/O driver, reconfiguring....237
I/Os, reconfiguring....237
Identification plate....64
IF ... THEN ... ENDIF....597
IF ... THEN ... ENDIF (inline form)....534
IN parameters....621
Increment....98
Incremental jogging....98
Indirect (method)....208
Industrial robot....23, 27
Info (menu item)....120
Inline forms, names....451
Inline forms, WorkVisual....451
Inputs, Automatic External....279
Inputs/outputs....703
Inputs/outputs, analog....111
Inputs/outputs, Automatic External....111, 278
Inputs/outputs, digital....109
Installation....723
Intended use....25
INTERIMENERGY....130
INTERN.ZIP....358, 359
Interrupt....625
Interrupt declaration....625
INTERRUPT DISABLE....629
INTERRUPT ENABLE....629
INTERRUPT OFF....628
INTERRUPT ON....628
Interrupt program....531, 625
Interrupts (menu item)....718
Interrupts, displaying....718
Introduction....21
INV_POS....664
INVERSE....665
Invisible, subprogram....392
IP addresses....229
ISTEP....377
IT security....48

J

Jog keys....62, 87, 93, 94
Jog mode....40, 45, 46
Jog override....92
Jogging, axis-specific....86, 93
Jogging, Cartesian....86, 94, 97
Jogging, external axes....104
Jogging, robot....86
Jump....596

K

Kernel system memory..... 121
Keyboard.....62
Keyboard key.....62
Keypad.....67
Keywords..... 546
Kinematic system, external..... 196
Kinematics group..... 66, 88
KLI, configuring.....229
KRCDiag.....720
KRL instructions, for SUB programs..... 702
KSS..... 23, 29
KUKA Customer Support.....120, 751
KUKA Line Interface, configuring..... 229
KUKA Service.....751
KUKA smartHMI..... 65
KUKA smartPAD.....29
KUKA smartPAD-2..... 29
KUKA System Software..... 23
KUKA.SecurityUpdate.....121

L

Labeling..... 42
Language.....78
Licenses.....22
LIN..... 556
LIN motion..... 453
LIN, motion type.....416
LIN_REL..... 559
Line break (menu item).....350
Line mark for mastering.....179
Linear motion..... 556, 559
Linear unit..... 27, 210
Load data.....213
Load data verification.....219
Logbook..... 713
Logbook, configuring..... 715
Long texts, exporting.....221
Long texts, importing.....221
LOOP ... ENDLOOP.....598
LOOP ... ENDLOOP (inline form).....538
Loss of mastering..... 168, 173, 179, 184
Low Voltage Directive.....28

M

Machine data.....51, 120, 122
Machine data, configuration.....148
Machinery Directive.....28
Main menu key.....62
Main menu, calling.....71
Maintenance.....56, 224
MAMES.....223
Manipulator..... 23, 27, 29
Manual mode.....55
Mastering..... 161
Mastering after maintenance work..... 177
Mastering marks.....164, 165
Mastering methods..... 162
Mastering position, A6.....179
Mastering, deleting..... 186
Mechanical end stops..... 41
MEMD.....163, 178
Memory capacities.....121
Message help..... 78
Message window..... 66
Messages, displaying help.....79
Micro Electronic Mastering Device.....163, 178
Minimizing KUKA smartHMI.....70
Misuse.....25
Mixed approximate positioning..... 642
Mode selector switch.....62
Modifying a logic instruction..... 543
Modifying a variable..... 112
Modifying coordinates.....503
Modifying motion parameters..... 502, 503
Modifying variables..... 114
Module..... 546
Monitoring time.....320
Monitoring, physical safeguards.....36
Monitoring, velocity.....40
Motherboard.....121
Motion conditions (window).....68
Motion programming, basic principles..... 415
Motion types..... 415
MSTEP.....377
Multi Submit.....704
Multiplication..... 649

N

Name, archive..... 122
Name, control PC.....121
Name, controller.....121, 122
Names.....546
Navigator.....340
Near range approximate positioning... 457, 562
Non-rejecting loop.....537, 607
Numeric entry, linear unit.....213

O

Odd parity.....286
Offset.....168, 172, 178, 183, 528
ON_ERROR_PROCEED.....599
Online documentation.....78
Online optimizing.....319
Open all FOLDs (menu item)..... 351
Open source.....22
Opening a program..... 345
Operating hours..... 121, 122
Operating hours meter..... 122
Operating mode after reconfiguration.....238
Operating mode after start.....73
Operating mode selection.....43, 45
Operating mode, changing.....84
Operation..... 61
Operator (user group)..... 84

Operator safety....35, 36, 44, 46, 69
Operator safety acknowledgement....151
Operator, geometric....650
Operators....32
Operators for bit operations....655
Operators for comparison operations....653
Operators, arithmetic....649
Operators, logic....654
Operators, priority....657
Option packages, backup....366, 367
Option packages, display....121
Option packages, restoring....366
Options....23, 27
Orientation behavior, SCIRC....439
Orientation control, LIN, CIRC....418
Orientation control, spline....436
OUT (inline form)....518
OUT parameters....621
Output, analog....526
Output, digital....518
Outputs, Automatic External....287
Overload....47
Override....92, 381
Override (menu item)....106
Overriding, power failure....75
Overview of the industrial robot....23
Overview, safety acceptance....137

P

Palletizing robot....200
Palletizing robots....160
Panic position....38
Parameters, transferring....621
Parity....286
Parity bit....286
Parity, even....286
Parity, odd....286
Password, changing....251
Paste....356
PATH....638
Peak values, collision detection..394, 396, 398
Performance Level....35
Peripheral contactor....54, 151, 159
Personal protective equipment....32
Personnel....31
PGNO_FBIT....281
PGNO_FBIT_REFL....289
PGNO_LENGTH....281
PGNO_PARITY....282
PGNO_REQ....291
PGNO_TYPE....279
PGNO_VALID....282
Pinning....361
Plant integrator....31
Point-to-point....415
Point-to-point motion....555, 558
Point correction, defining limits....268
Point, global....510
Positionally accurate robot, checking activation....159
Positioner....27, 193
Power failure, overriding....75
Power failure, wait time....75
PPE....32
Pre-mastering position....164, 166
Pressure Equipment Directive....58
Preventive maintenance work....57
Priority....626, 635, 639
Probe....163
Processor....121
Product description....23
PROFIenergy....123
PROFINET interface....231
Program execution....377
Program execution control....594
Program lines, deleting....352
Program override....381
Program run mode, selecting....377
Program run modes....377
Program, canceling....345
Program, closing....346
Program, creating....339
Program, editing....352
Program, opening....345
Program, selecting....345
Program, starting....382, 383
Program, stopping....383, 386
Project management (window)....363
Project, activating....361
Project, inactive....364
Projects, backup....366, 367
Projects, restoring....366
Properties, file or folder....342
Protective equipment....40
PTP....555
PTP motion....452
PTP spline block....468, 469, 566
PTP, motion type....415
PTP_REL....558
PTP_SPLINE ... ENDSPLINE....566
PUBLIC....550
Pulse....519, 613
PULSE (inline form)....519
Pulse, path-related....525

R

RAM....121
RDC data, restoring....370
RDC, data backup....123, 366, 367
RDC, exchange....177
Re-teaching....503
Reaction distance....28
READ_CM_SETTING....411
Recommissioning....49, 137
Reference mastering....177
Reference system for jog keys....66
Reference system for Space Mouse....66

REFLECT_PROG_NR....280
Rejecting loop....535, 610
Release device....42
Renaming a file....339
Renaming a folder....339
Repair....56
REPEAT ... UNTIL....607
REPEAT ... UNTIL (inline form)....537
Replace....356
RESET (command)....708
RESET (state)....691
RESET_TORQUE_LIMITS....308, 702
Resetting a program....385
Resetting, submit interpreters....695
Restoration, option packages....366
Restoration, projects....366
Restoring data....360
Restoring RDC data....370
RESUME....632
Reteaching, defining limits....268
RETURN....620
Rights management....146, 252
ROB_STOP....667
ROB_STOP_RELEASE....667, 669
Robot controller....23, 27
Robot controller, switching on....137
Robot data (menu item)....121
Robot interpreter....691
ROBROOT coordinate system....85
Root point (method)....209
RUN (command)....708
Runtime variable....549

S

Safe operational stop....29
Safeguards, external....42
Safety....27
Safety acceptance, overview....137
Safety acceptance, precondition....729
Safety configuration, exporting....246
Safety configuration, importing....246
Safety controller....35
Safety functions....34, 44, 45
Safety functions, overview....34
Safety instructions....21
Safety maintenance (user group)....83
Safety measures....46
Safety options....30
Safety recovery (user group)....83
Safety STOP 0....29
Safety STOP 1....30
Safety STOP 2....30
Safety stop 0....29
Safety stop 1....30
Safety STOP 1 – Drive Ramp Stop....30
Safety STOP 1 DRS....30
Safety stop 2....30
Safety updates....121
Safety zone....29, 32
Safety, general....27
SAVE_CM_MAX_SETTING....411
SaveMax (inline form)....406
SCIRC....569
SCIRC motion, programming....483
SCIRC_REL....573
Screenshot, smartPAD....78
SEC....609
Selected section (marked region)....356
Selecting a program....345
Selecting the base....93
Selecting the tool....93
Selecting, range in programSelecting, line in program....352
SEMD....163, 168
Serial number....122
Service life....29, 120
SET_BRAKE_DELAY....669
SET_TORQUE_LIMITS....305, 702
SetDefault (inline form)....407
SetLimits (inline form)....407
SGTLCRC.xml....246
Shutdown (menu item)....73, 77
Shutting down the robot controller....73
SID....121
SIGNAL....617
Signal declaration, information....662
Signal diagrams....296
Signals, brake test....324, 327
Simulation....56
Sine (mathematical function)....658
Single (menu item)....112, 114
Single point of control....58
Single Submit....704
Singularities....448
Singularity, CP spline....436, 438
Singularity, LIN/CIRC....418
SLIN....567
SLIN motion, programming....481
SLIN_REL....572
smartHMI....65
smartPAD....30, 47, 61
smartPAD-2....30, 61
Soft axes....302
Software....23, 27
Software limit switches....40, 45, 46, 186
Software limit switches, modifying....186
Space Mouse....62, 87, 94, 96, 97
Special characters....451
SPL....570
SPL_REL....575
SPLINE ... ENDSPLINE....565
Spline block, programming....468, 469
Spline segment....421
Spline, motion type....421
SPOC....58
SPS.SUB....691, 700
sps.sub, editing....700
SPTP....570
SPTP motion, programming....486

SPTP_REL....576
Square root (mathematical function)....658
SRC....546
SREAD....649
Stamp....353
Standard Electronic Mastering Device 163, 168
Standstill monitoring....239
Start-up....49, 137
Start-up mode....54
Start-up wizard....148
Start backwards key....62
Start key....62, 63
Start type, KSS....72
Start types....76
Starting a program, automatic....383
Starting a program, manual....382
Starting Automatic External mode....385
Starting the KSS....70
Starting, submit interpreter....696
Starting, submit interpreters....693, 695, 696
Starting, system submit interpreter....696
States, submit interpreter....691
Status....444, 445
Status bar....66, 67, 692
Status indicator....692
Status keys....62
Status line....340
STEP....595
STOP (command)....708
STOP (state)....691
STOP 0....28, 31
STOP 1....28, 31
STOP 2....28, 31
Stop category 0....31
Stop category 1....31
Stop category 2....31
Stop category 1, Drive Ramp Stop....31
STOP key....62
Stop reactions....33, 34
STOP WHEN PATH....586
STOP 1 - DRS....31
Stop, robot....667
Stopping a program....383, 386
Stopping distance....28, 33
Stopping, robot....631
Stopping, submit interpreters....694, 695
Storage....58
StrAdd....686
StrClear....686
StrComp....688
StrCopy....689
StrDeclLen....685
StrFind....687
String variable length after initialization....685
String variable length in declaration....685
String variables....674
String variables, comparing contents....688
String variables, copying....689
String variables, deleting contents....686
String variables, extending....686
String variables, searching....687
StrLen....685
StrToAXIS....674
StrToBOOL....681
StrToE3AXIS....675
StrToE3POS....679
StrToE6AXIS....676
StrToE6POS....680
StrToFRAME....677
StrToINT....682
StrToPOS....678
StrToREAL....683
StrToSTRING....683
STRUC....554
Structure type....554
SUB program, creating....699
Submit (template)....700
Submit interpreter....68, 691
Subprogram, calling....618
Subprogram, invisible....392
Subprograms, of a SUB program....702
Subtraction....649
Supplementary load data (menu item)....216, 217
Supplementary load data, importing....217
supplementary load data, numeric entry....216
Support request....751
SWITCH ... CASE ... ENDSWITCH....607
SWITCH ... CASE ... ENDSWITCH (inline form)....541
Switching action, path-related....520
Switching on the robot controller....70
SWRITE....649
Symbols....545
SYN OUT....520
SYN PULSE (inline form)....525
System integrator....28, 31, 32
System requirements....25, 723
System submit....691
System submit interpreter....691
System variables....391
System variables, submit interpreter....703

T

T1 (operating mode)....31, 45
T2 (operating mode)....31, 45
Tangent (mathematical function)....658
TCP....190
Teach mode, collision detection..400, 402, 406
Teach pendant....23, 27
Teaching....503
Technology packages....451, 546
Technology packages, display....121
Terms, safety....28
Time block....578
TIME_BLOCK....578
Timer, programming (inline form)....528
Timers, displaying....119
TOOL....190

Tool Center Point....190
TOOL coordinate system....85, 190
Tool direction....189
TORQMON....393
Torque mode, diagnosis....310
Torque mode, examples....304, 313
Torque mode, overview....302
Touch screen....67
Trace (jogging option)....89
Trademarks....22
Training....21
Transfer motion....461
Transforming coordinates....503
Transportation....49
TRIGGER....634, 638
Turn....444, 447
Turn-tilt table....27, 193
Type, robot....120
Type, robot controller....120

U

Uncommenting....354
Unmastering....186
Update....725
US2....54, 151, 159
USB sticks....25
UseDataSet (inline form)....403
User....29, 32
User (user group)....83
User group, changing....82
User groups....83
User interface....65
User rights....251

V

Variable display....112
Variable overview, configuring....249
VARSTATE....114, 672
VECTORMOVEOFF....702
VECTORMOVEON....702
Velocity....92, 381
Velocity monitoring....40
Version, kernel system....120
Version, operating system....121
Version, robot controller....120
Version, user interface....120
Voltage....527, 528
VSS....31

W

WAIT....609
WAIT (inline form)....516
WAIT FOR....609
WAIT FOR (inline form)....517
Wait function, signal-dependent....517
WAIT SEC....609
Wait time....516, 609
Wait time, power failure....75
Warm-up....269
Warnings....21
WHILE ... ENDWHILE....610
WHILE ... ENDWHILE (inline form)....535
Windows interface....70, 229, 231
WITH (permissible system variables)....577
Workspace....28, 32
Workspaces, axis-specific....257
Workspaces, bypassing monitoring....105
Workspaces, Cartesian....257
Workspaces, mode....262, 266
WorkVisual, inline forms....451
WORLD coordinate system....85
Wrist root point....263

X

XML export....246
XML import....246
XYZ (method)....203
XYZ 3-point (method)....202
XYZ 4-point (method)....200
XYZ Reference (method)....202