

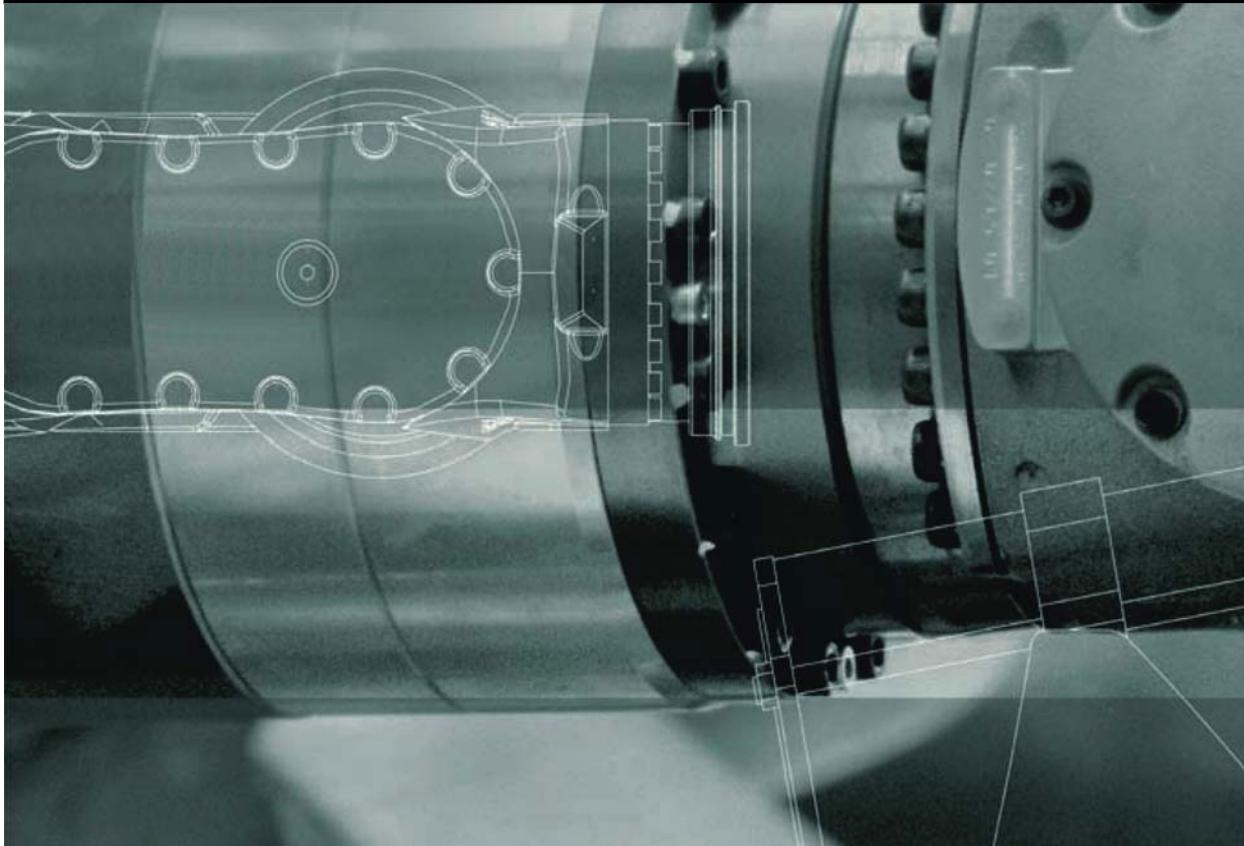
# KUKA

KUKA System Software

KUKA Deutschland GmbH

## KUKA System Software 8.3

**Operating and Programming Instructions for System Integrators**



Issued: 18.04.2018

Version: KSS 8.3 SI V7

© Copyright 2018

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 8.3 SI (PDF) en

Book structure: KSS 8.3 SI V7.4

Version: KSS 8.3 SI V7

## Contents

<b>1</b>	<b>Introduction</b>	15
1.1	Target group	15
1.2	Industrial robot documentation	15
1.3	Representation of warnings and notes	15
1.4	Trademarks	16
<b>2</b>	<b>Product description</b>	17
2.1	Overview of the industrial robot	17
2.2	Overview of KUKA System Software (KSS)	17
2.3	System requirements	18
2.4	Intended use of the KUKA System Software	18
2.5	KUKA USB sticks	18
<b>3</b>	<b>Safety</b>	19
3.1	General	19
3.1.1	Liability	19
3.1.2	Intended use of the industrial robot	19
3.1.3	EC declaration of conformity and declaration of incorporation	20
3.1.4	Terms used	20
3.2	Personnel	22
3.3	Workspace, safety zone and danger zone	23
3.3.1	Determining stopping distances	23
3.4	Triggers for stop reactions	24
3.5	Safety functions	24
3.5.1	Overview of the safety functions	24
3.5.2	Safety controller	25
3.5.3	Selecting the operating mode	25
3.5.4	“Operator safety” signal	26
3.5.5	EMERGENCY STOP device	27
3.5.6	Logging off from the higher-level safety controller	27
3.5.7	External EMERGENCY STOP device	28
3.5.8	Enabling device	28
3.5.9	External enabling device	29
3.5.10	External safe operational stop	29
3.5.11	External safety stop 1 and external safety stop 2	29
3.5.12	Velocity monitoring in T1	29
3.6	Additional protective equipment	29
3.6.1	Jog mode	29
3.6.2	Software limit switches	30
3.6.3	Mechanical end stops	30
3.6.4	Mechanical axis limitation (optional)	30
3.6.5	Options for moving the manipulator without drive energy	30
3.6.6	Labeling on the industrial robot	31
3.6.7	External safeguards	31
3.7	Overview of operating modes and safety functions	32
3.8	Safety measures	32
3.8.1	General safety measures	32

3.8.2	Transportation .....	33
3.8.3	Start-up and recommissioning .....	34
3.8.3.1	Checking machine data and safety configuration .....	35
3.8.3.2	Start-up mode .....	37
3.8.4	Manual mode .....	38
3.8.5	Simulation .....	39
3.8.6	Automatic mode .....	39
3.8.7	Maintenance and repair .....	39
3.8.8	Decommissioning, storage and disposal .....	41
3.8.9	Safety measures for “single point of control” .....	41
3.9	Applied norms and regulations .....	42
<b>4</b>	<b>Operation .....</b>	<b>45</b>
4.1	KUKA smartPAD teach pendant .....	45
4.1.1	Front view .....	45
4.1.2	Rear view .....	47
4.1.3	Disconnecting and connecting the smartPAD .....	48
4.2	KUKA smartHMI user interface .....	49
4.2.1	Keypad .....	50
4.2.2	Status bar .....	51
4.2.3	<b>Drives</b> status indicator and <b>Motion conditions</b> window .....	52
4.2.4	Minimizing KUKA smartHMI (displaying Windows interface) .....	53
4.3	Switching on the robot controller and starting the KSS .....	54
4.4	Calling the main menu .....	54
4.5	Defining the start type for KSS .....	55
4.6	Exiting or restarting KSS .....	55
4.6.1	Shutting down after power failure .....	58
4.7	Switching drives on/off .....	58
4.8	Switching the robot controller off .....	58
4.9	Setting the user interface language .....	59
4.10	Creating a screenshot on the smartPAD .....	59
4.11	Online documentation and help for messages .....	59
4.11.1	Calling online documentation .....	59
4.11.2	Calling help for the messages .....	60
4.12	Changing user group .....	62
4.13	Changing operating mode .....	63
4.14	Coordinate systems .....	64
4.15	Jogging the robot .....	65
4.15.1	“ <b>Jog options</b> ” window .....	66
4.15.1.1	“ <b>General</b> ” tab .....	67
4.15.1.2	“ <b>Keys</b> ” tab .....	67
4.15.1.3	“ <b>Mouse</b> ” tab .....	68
4.15.1.4	“ <b>KCP pos.</b> ” tab .....	69
4.15.1.5	“ <b>Cur. tool/base</b> ” tab .....	70
4.15.2	Activating the jog mode .....	70
4.15.3	Setting the jog override (HOV) .....	70
4.15.4	Selecting the tool and base .....	71
4.15.5	Axis-specific jogging with the jog keys .....	71
4.15.6	Cartesian jogging with the jog keys .....	71

4.15.7	Configuring the Space Mouse .....	72
4.15.8	Defining the alignment of the Space Mouse .....	74
4.15.9	Cartesian jogging with the Space Mouse .....	75
4.15.10	Incremental jogging .....	75
4.16	Jogging external axes .....	76
4.17	Bypassing workspace monitoring .....	76
4.18	Display functions .....	77
4.18.1	Measuring and displaying energy consumption .....	77
4.18.2	Displaying the actual position .....	79
4.18.3	Displaying digital inputs/outputs .....	79
4.18.4	Displaying analog inputs/outputs .....	81
4.18.5	Displaying inputs/outputs for Automatic External .....	81
4.18.6	Displaying and modifying the value of a variable .....	82
4.18.7	Displaying the state of a variable .....	83
4.18.8	Displaying the variable overview and modifying variables .....	84
4.18.9	Displaying cyclical flags .....	85
4.18.10	Displaying flags .....	86
4.18.11	Displaying counters .....	87
4.18.12	Displaying timers .....	88
4.18.13	Displaying calibration data .....	89
4.18.14	Displaying information about the robot and robot controller .....	90
4.18.15	Displaying/editing robot data .....	90
4.19	Resetting safe input/output errors .....	92
4.20	Displaying the battery state .....	93
<b>5</b>	<b>Start-up and recommissioning .....</b>	<b>95</b>
5.1	Protecting the Windows system .....	95
5.2	Changing the password for a Windows system .....	95
5.2.1	Changing the password: return values .....	96
5.3	Start-up wizard .....	97
5.4	Modifying the machine data configuration .....	97
5.5	Defining hardware options .....	98
5.6	Changing the safety ID of the PROFINET device .....	99
5.7	Jogging the robot without a higher-level safety controller .....	100
5.8	Checking the activation of the positionally accurate robot model .....	101
5.9	Activating palletizing mode .....	102
5.10	Mastering .....	103
5.10.1	Mastering methods .....	104
5.10.2	Moving axes to the pre-mastering position using mastering marks .....	105
5.10.3	Moving axes to the pre-mastering position using the probe .....	107
5.10.4	Mastering LEDs .....	108
5.10.5	Mastering with the SEMD .....	109
5.10.5.1	First mastering (with SEMD) .....	109
5.10.5.2	Teach offset (with SEMD) .....	112
5.10.5.3	Check load mastering with offset (with SEMD) .....	113
5.10.6	Mastering with the dial gauge .....	114
5.10.7	Mastering external axes .....	115
5.10.8	Reference mastering .....	115
5.10.9	Mastering with the MEMD and mark .....	116

5.10.9.1	Moving A6 to the mastering position (with line mark) .....	117
5.10.9.2	First mastering (with MEMD) .....	118
5.10.9.3	Teach offset (with MEMD) .....	121
5.10.9.4	Check load mastering with offset (with MEMD) .....	122
5.10.10	Manually unmastering axes .....	123
5.11	Modifying software limit switches .....	123
5.12	Calibration .....	126
5.12.1	Defining the tool direction .....	126
5.12.2	Tool calibration .....	126
5.12.2.1	TCP calibration: XYZ 4-point method .....	128
5.12.2.2	TCP calibration: XYZ Reference method .....	129
5.12.2.3	Defining the orientation: ABC World method .....	130
5.12.2.4	Defining the orientation: ABC 2-point method .....	131
5.12.2.5	Numeric input .....	132
5.12.3	Base calibration .....	133
5.12.3.1	Base calibration: 3-point method .....	133
5.12.3.2	Base calibration: indirect method .....	135
5.12.3.3	Entering the base numerically .....	136
5.12.4	Fixed tool calibration .....	136
5.12.4.1	Calibrating an external TCP .....	136
5.12.4.2	Entering the external TCP numerically .....	138
5.12.4.3	Workpiece calibration: direct method .....	138
5.12.4.4	Workpiece calibration: indirect method .....	140
5.12.5	Renaming the tool/base .....	140
5.12.6	Linear unit .....	141
5.12.6.1	Checking whether the linear unit needs to be calibrated .....	141
5.12.6.2	Calibrating the linear unit .....	142
5.12.6.3	Entering the linear unit numerically .....	143
5.12.7	Calibrating an external kinematic system .....	144
5.12.7.1	Calibrating the root point .....	144
5.12.7.2	Entering the root point numerically .....	146
5.12.7.3	Calibrating a workpiece base .....	146
5.12.7.4	Entering the workpiece base numerically .....	148
5.12.7.5	Calibrating an external tool .....	148
5.12.7.6	Entering the external tool numerically .....	149
5.13	Load data .....	150
5.13.1	Checking loads with KUKA.Load .....	150
5.13.2	Calculating payloads with KUKA.LoadDataDetermination .....	150
5.13.3	Entering payload data .....	150
5.13.4	Entering supplementary load data .....	151
5.13.5	Online load data check (OLDC) .....	151
5.14	Exporting/importing long texts .....	153
5.15	Maintenance handbook .....	155
5.15.1	Logging maintenance .....	156
5.15.2	Displaying a maintenance log .....	157
<b>6</b>	<b>Configuration .....</b>	<b>159</b>
6.1	Configuring the KUKA Line Interface (KLI) .....	159
6.1.1	Configuring the Windows interface (without field bus) .....	159
6.1.2	Configuring the field bus interface and creating the Windows interface .....	161

6.1.3	Displaying ports of the Windows interface or enabling an additional port .....	162
6.1.4	Displaying or modifying filters .....	163
6.1.5	Error display in the Address and Subnet boxes .....	163
6.2	Displaying the subnet configuration of the robot controller .....	164
6.3	Reconfiguring the I/O driver .....	165
6.4	Configuring safe axis monitoring functions .....	165
6.4.1	Parameter <b>Braking time</b> .....	166
6.4.2	Parameter <b>Maximum velocity T1</b> for couplable axes .....	168
6.4.3	Checking the limits for the maximum axis velocity in T1 mode .....	169
6.5	Checking the values for the safe axis monitoring functions .....	170
6.6	Checking the safety configuration of the robot controller .....	171
6.7	Checksum of the safety configuration .....	172
6.8	Exporting the safety configuration (XML export) .....	172
6.9	Checking via PLC whether the safety configuration has been changed .....	172
6.10	Configuring the variable overview .....	174
6.11	Changing the password .....	176
6.12	Energy saving mode (\$ECO_LEVEL) .....	176
6.13	Configuring workspaces .....	177
6.13.1	Configuring Cartesian workspaces .....	177
6.13.2	Configuring axis-specific workspaces .....	180
6.13.3	Mode for workspaces .....	182
6.14	Defining limits for reteaching .....	182
6.15	Warm-up .....	183
6.15.1	Warm-up sequence .....	184
6.15.2	Configuring warm-up .....	185
6.16	Collision detection .....	186
6.16.1	Calculating the tolerance range and activating collision detection .....	188
6.16.2	Defining an offset for the tolerance range .....	189
6.16.3	Option window “Collision detection” .....	190
6.16.4	Editing the program <b>tm_useraction</b> .....	191
6.16.5	Torque monitoring .....	192
6.16.5.1	Determining values for torque monitoring .....	192
6.16.5.2	Programming torque monitoring .....	193
6.17	Defining calibration tolerances .....	193
6.18	Configuring backward motion .....	194
6.19	Configuring Automatic External .....	195
6.19.1	Configuring CELL.SRC .....	196
6.19.2	Configuring Automatic External inputs/outputs .....	197
6.19.2.1	Automatic External inputs .....	199
6.19.2.2	Odd / even parity .....	201
6.19.2.3	Automatic External outputs .....	202
6.19.3	Transmitting error numbers to the higher-level controller .....	204
6.19.4	Signal diagrams .....	206
6.20	Torque mode .....	212
6.20.1	Overview of torque mode .....	212
6.20.1.1	Using torque mode .....	212
6.20.1.2	Robot program example: setting A1 to “soft” in both directions .....	214
6.20.2	Activating torque mode: SET_TORQUE_LIMITS() .....	215
6.20.3	Deactivating torque mode: RESET_TORQUE_LIMITS() .....	218

6.20.4	Interpreter specifics .....	218
6.20.5	Diagnostic variables for torque mode .....	219
6.20.5.1	\$TORQUE_AXIS_ACT .....	219
6.20.5.2	\$TORQUE_AXIS_MAX_0 .....	220
6.20.5.3	\$TORQUE_AXIS_MAX .....	220
6.20.5.4	\$TORQUE_AXIS_LIMITS .....	220
6.20.5.5	\$HOLDING_TORQUE .....	221
6.20.5.6	Comparison: \$TORQUE_AXIS_ACT and \$HOLDING_TORQUE .....	221
6.20.6	Other examples .....	222
6.20.6.1	Robot program: setting axis to “soft” in both directions .....	222
6.20.6.2	Robot program: avoiding damage in the event of collisions .....	223
6.20.6.3	Robot program: torque mode in the interrupt .....	224
6.20.6.4	Robot program: servo gun builds up pressure .....	225
6.20.6.5	Submit program: servo gun builds up pressure .....	226
6.21	<b>Event planner:</b> Configuring a data comparison .....	227
6.22	Brake test .....	227
6.22.1	Overview of the brake test .....	227
6.22.2	Activating the brake test .....	229
6.22.3	Programs for the brake test .....	229
6.22.4	Configuring input and output signals for the brake test .....	230
6.22.4.1	Signal diagram of the brake test – examples .....	231
6.22.5	Teaching positions for the brake test .....	233
6.22.6	Performing a manual brake test .....	233
6.22.7	Checking that the brake test is functioning correctly .....	234
<b>7</b>	<b>Program and project management</b> .....	237
7.1	Creating a new program .....	237
7.2	Creating a new folder .....	237
7.3	Renaming a file or folder .....	237
7.4	<b>Navigator</b> file manager .....	238
7.4.1	Selecting filters .....	239
7.4.2	Displaying or modifying properties of files and folders .....	239
7.5	Selecting or opening a program .....	243
7.5.1	Selecting and deselecting a program .....	243
7.5.2	Opening a program .....	244
7.5.3	Toggling between the Navigator and the program .....	245
7.6	Structure of a KRL program .....	245
7.6.1	HOME position .....	246
7.7	Displaying/hiding program sections .....	247
7.7.1	Displaying/hiding the DEF line .....	247
7.7.2	Activating detail view .....	247
7.7.3	Activating/deactivating the line break function .....	247
7.7.4	Displaying Folds .....	248
7.8	Editing programs .....	249
7.8.1	Inserting a comment or stamp .....	250
7.8.2	Creating folds .....	251
7.8.3	Selecting a range .....	252
7.8.4	Deleting program lines .....	252
7.8.5	Additional editing functions .....	252

7.9	Printing a program .....	253
7.10	Archiving and restoring data .....	253
7.10.1	Archiving overview .....	253
7.10.2	Archiving to a USB stick .....	254
7.10.3	Archiving on the network .....	255
7.10.4	Archiving the logbook .....	255
7.10.5	Restoring data .....	256
7.11	Project management .....	256
7.11.1	Pinning a project on the robot controller .....	256
7.11.2	Activating a project .....	257
7.11.3	<b>Project management</b> window .....	258
7.12	Backup Manager .....	260
7.12.1	Overview of Backup Manager .....	260
7.12.2	Backing up projects, option packages and RDC data manually .....	261
7.12.3	Manually restoring projects and option packages .....	262
7.12.4	Restoring RDC data manually .....	263
7.12.5	Configuring Backup Manager .....	264
7.12.5.1	<b>Backup configuration</b> tab .....	265
7.12.5.2	“ <b>Signal interface</b> ” tab .....	267
<b>8</b>	<b>Program execution</b> .....	269
8.1	Selecting the program run mode .....	269
8.2	Program run modes .....	269
8.3	Advance run .....	270
8.4	Block pointer .....	270
8.5	Setting the program override (POV) .....	273
8.6	Robot interpreter status indicator .....	273
8.7	Starting a program forwards (manual) .....	273
8.8	Starting a program forwards (automatic) .....	274
8.9	Carrying out a block selection .....	274
8.10	Resetting a program .....	275
8.11	Starting Automatic External mode .....	275
8.12	Backward motion using the Start backwards key .....	276
8.12.1	Executing motions backwards (using the “Start backwards” key) .....	276
8.12.2	Functional principle and characteristics of backward motion .....	276
8.12.2.1	Response in the case of subprograms .....	277
8.12.2.2	Approximate positioning response .....	278
8.12.2.3	Response in the case of weave motions .....	279
8.12.2.4	Switching from backwards to forwards .....	280
8.12.3	System variables with changed meaning .....	280
<b>9</b>	<b>Basic principles of motion programming</b> .....	283
9.1	Overview of motion types .....	283
9.2	Motion type PTP .....	283
9.3	Motion type LIN .....	284
9.4	Motion type CIRC .....	284
9.5	Approximate positioning .....	285
9.6	Orientation control LIN, CIRC .....	286
9.6.1	Combinations of \$ORI_TYPE and \$CIRC_TYPE .....	287

9.7	Spline motion type .....	289
9.7.1	Velocity profile for spline motions .....	291
9.7.2	Block selection with spline motions .....	292
9.7.3	Modifications to spline blocks .....	293
9.7.4	Approximation of spline motions .....	296
9.7.5	Replacing an approximated CP motion with a spline block .....	296
9.7.5.1	SLIN-SPL-SLIN transition .....	299
9.8	Orientation control for CP spline motions .....	300
9.8.1	SCIRC: reference system for the orientation control .....	302
9.8.2	SCIRC: orientation behavior .....	302
9.8.2.1	SCIRC: Orientation behavior – example: auxiliary point .....	303
9.8.2.2	SCIRC: Orientation behavior – example: end point .....	305
9.9	Circular angle .....	306
9.10	Status and Turn .....	307
9.10.1	Status .....	307
9.10.2	Turn .....	310
9.11	Singularities .....	311
<b>10</b>	<b>Programming for user group “User” (inline forms) .....</b>	<b>313</b>
10.1	Instructions for programming .....	313
10.2	Names in inline forms .....	313
10.3	Programming PTP, LIN and CIRC motions .....	313
10.3.1	Programming a PTP motion .....	313
10.3.2	Inline form “PTP” .....	314
10.3.3	Programming a LIN motion .....	314
10.3.4	Inline form “LIN” .....	315
10.3.5	Programming a CIRC motion .....	315
10.3.6	Inline form “CIRC” .....	316
10.3.7	Option window “ <b>Frames</b> ” .....	316
10.3.8	Option window “ <b>Motion parameters</b> ” (LIN, CIRC, PTP) .....	317
10.4	Programming spline motions .....	318
10.4.1	Programming tips for spline motions .....	318
10.4.2	Programming a spline block .....	319
10.4.2.1	Inline form for CP spline block .....	320
10.4.2.2	Inline form “ <b>PTP SPLINE block</b> ” .....	321
10.4.2.3	Option window “ <b>Frames</b> ” (CP and PTP spline block) .....	321
10.4.2.4	Option window “ <b>Motion parameters</b> ” (CP spline block) .....	322
10.4.2.5	Option window “ <b>Motion parameters</b> ” (PTP spline block) .....	323
10.4.3	Programming segments for a spline block .....	323
10.4.3.1	Programming an SPL or SLIN segment .....	323
10.4.3.2	Programming an SCIRC segment .....	323
10.4.3.3	Inline form for CP spline segment .....	324
10.4.3.4	Programming an SPTP segment .....	325
10.4.3.5	Inline form for SPTP segment .....	325
10.4.3.6	Option window “ <b>Frames</b> ” (CP and PTP spline segments) .....	326
10.4.3.7	Option window “ <b>Motion parameters</b> ” (CP spline segment) .....	327
10.4.3.8	Option window “ <b>Motion parameters</b> ” (SPTP) .....	328
10.4.3.9	Option window “ <b>Logic parameters</b> ” .....	328
10.4.3.10	Teaching the shift in space for logic parameters .....	331
10.4.4	Programming individual spline motions .....	332

10.4.4.1	Programming an individual SLIN motion .....	332
10.4.4.2	Inline form “SLIN” .....	332
10.4.4.3	Option window “ <b>Motion parameters</b> ” (SLIN) .....	333
10.4.4.4	Programming an individual SCIRC motion .....	334
10.4.4.5	Inline form “SCIRC” .....	334
10.4.4.6	Option window “ <b>Motion parameters</b> ” (SCIRC) .....	335
10.4.4.7	Programming an individual SPTP motion .....	336
10.4.4.8	Inline form “ <b>SPTP</b> ” .....	336
10.4.5	Conditional stop .....	337
10.4.5.1	Inline form “ <b>Spline Stop Condition</b> ” .....	337
10.4.5.2	Stop condition: example and braking characteristics .....	339
10.4.6	Constant velocity range in the CP spline block .....	340
10.4.6.1	Block selection to the constant velocity range .....	341
10.4.6.2	Maximum limits .....	341
10.5	Displaying the distance between points .....	341
10.6	Modifying programmed motions .....	342
10.6.1	Modifying motion parameters .....	342
10.6.2	Modifying blocks of motion parameters .....	342
10.6.3	Re-teaching a point .....	342
10.6.4	Transforming blocks of coordinates .....	343
10.6.4.1	“ <b>Axis mirroring</b> ” window .....	346
10.6.4.2	“ <b>Transform - Axis Specific</b> ” window .....	347
10.6.4.3	“ <b>Transform - Cartesian Base</b> ” window .....	348
10.7	Programming logic instructions .....	349
10.7.1	Inputs/outputs .....	349
10.7.2	Setting a digital output - OUT .....	349
10.7.3	Inline form “OUT” .....	349
10.7.4	Setting a pulse output - PULSE .....	350
10.7.5	Inline form “PULSE” .....	350
10.7.6	Setting an analog output - ANOUT .....	351
10.7.7	Inline form “ANOUT” (static) .....	351
10.7.8	Inline form “ANOUT” (dynamic) .....	351
10.7.9	Programming a wait time - WAIT .....	352
10.7.10	Inline form “WAIT” .....	352
10.7.11	Programming a signal-dependent wait function - WAITFOR .....	352
10.7.12	Inline form “WAITFOR” .....	353
10.7.13	Switching on the path - SYN OUT .....	354
10.7.14	Inline form “SYN OUT”, option “START/END” .....	354
10.7.15	Inline form “SYN OUT”, option “PATH” .....	357
10.7.16	Setting a pulse on the path - SYN PULSE .....	359
10.7.17	Inline form “SYN PULSE” .....	359
10.7.18	Modifying a logic instruction .....	360
11	<b>Programming for user group (KRL syntax)</b> .....	361
11.1	Instructions for programming .....	361
11.2	Overview of KRL syntax .....	361
11.3	Symbols and fonts .....	364
11.4	Important KRL terms .....	364
11.4.1	SRC files and DAT files .....	364
11.4.2	Naming conventions and keywords .....	364

11.4.3	Data types .....	366
11.4.4	Areas of validity .....	367
11.4.4.1	Making subprograms, functions and interrupts available globally .....	367
11.4.4.2	Making variables, constants, signals and user data types available globally ..	368
11.4.5	Constants .....	369
11.5	Variables and declarations .....	369
11.5.1	DECL: declaring variables, arrays and constants .....	369
11.5.2	ENUM: defining an enumeration type .....	370
11.5.3	STRUC: defining a structure type .....	371
11.6	Motion programming: PTP, LIN, CIRC .....	373
11.6.1	PTP: motion programming .....	373
11.6.2	LIN: motion programming .....	373
11.6.3	CIRC: motion programming .....	374
11.6.4	PTP_REL: relative motion programming .....	375
11.6.5	LIN_REL: relative motion programming .....	376
11.6.6	CIRC_REL: relative motion programming .....	377
11.6.7	Approximation parameters for PTP, LIN CIRC and ...._REL .....	378
11.6.8	REL motions for infinitely rotating axes .....	379
11.7	Motion programming: spline .....	380
11.7.1	SPLINE ... ENDSPLINE: programming a CP spline block .....	380
11.7.2	PTP_SPLINE ... ENDSPLINE: programming a PTP spline block .....	382
11.7.3	SLIN: motion programming .....	383
11.7.4	SCIRC: motion programming .....	383
11.7.5	SPL: motion programming .....	384
11.7.6	SPTP: motion programming .....	385
11.7.7	SLIN_REL: relative motion programming .....	386
11.7.8	SCIRC_REL: relative motion programming .....	387
11.7.9	SPL_REL: relative motion programming .....	388
11.7.10	SPTP_REL: relative motion programming .....	389
11.7.11	System variables for WITH .....	390
11.7.12	TIME_BLOCK: programming a time block for spline .....	392
11.7.13	CONST_VEL: programming a constant velocity range for spline motions .....	395
11.7.13.1	System variables for CONST_VEL .....	396
11.7.14	STOP WHEN PATH: programming a conditional stop for spline .....	397
11.7.15	\$EX_AX_IGNORE .....	398
11.8	Program execution control .....	399
11.8.1	CONTINUE: preventing an advance run stop .....	399
11.8.2	EXIT: exiting a loop .....	400
11.8.3	FOR ... TO ... ENDFOR: programming a counting loop .....	400
11.8.4	GOTO: jump to position in the program .....	401
11.8.5	HALT: stopping a program .....	402
11.8.6	IF ... THEN ... ENDIF: programming a conditional branch .....	402
11.8.7	LOOP ... ENDLOOP: programming an endless loop .....	403
11.8.8	ON_ERROR_PROCEED: suppressing a runtime error message .....	403
11.8.8.1	\$ERR .....	404
11.8.8.2	Examples of \$ERR, ON_ERROR_PROCEED and ERR_RAISE() .....	406
11.8.9	REPEAT ... UNTIL: programming a post-test loop .....	408
11.8.10	SWITCH ... CASE ... END SWITCH: programming a multiple branch .....	409
11.8.11	WAIT FOR ... : waiting until a condition has been met .....	410

11.8.12 WAIT SEC ... : programming a wait time .....	411
11.8.13 WHILE ... ENDWHILE: programming a rejecting loop .....	411
11.9 Inputs/outputs .....	412
11.9.1 ANIN: cyclically reading an analog input .....	412
11.9.2 ANOUT: writing cyclically to an analog output .....	413
11.9.3 PULSE: setting a pulse output .....	414
11.9.4 SIGNAL: signal declaration for inputs/outputs .....	418
11.10 Subprograms and functions .....	419
11.10.1 Calling a subprogram .....	419
11.10.2 Calling a function .....	419
11.10.3 DEFFCT ... ENDFCT: defining a function .....	420
11.10.4 RETURN: jump back to the calling program .....	420
11.10.5 Transferring parameters to a subprogram or function .....	421
11.10.6 Transferring a parameter to a different data type .....	425
11.11 Interrupt programming .....	425
11.11.1 INTERRUPT ... DECL ... WHEN ... DO ... : declaring an interrupt .....	425
11.11.2 INTERRUPT ON/OFF: activating or deactivating an interrupt .....	428
11.11.3 INTERRUPT DISABLE/ENABLE: disabling or enabling an interrupt .....	429
11.11.4 BRAKE: stopping the robot from an interrupt program .....	432
11.11.5 RESUME: aborting interrupt programs .....	433
11.12 Path-related switching actions (=Trigger) .....	435
11.12.1 TRIGGER WHEN DISTANCE .....	435
11.12.2 TRIGGER WHEN PATH .....	438
11.12.2.1 Reference point for approximate positioning – overview .....	442
11.12.2.2 Reference point for homogenous approximate positioning .....	443
11.12.2.3 Reference point for mixed approximate positioning (spline) .....	444
11.12.2.4 Reference point for mixed approximate positioning (LIN/CIRC/PTP) .....	445
11.12.3 Constraints for functions in the trigger .....	445
11.12.4 Useful system variables for working with PATH triggers .....	446
11.12.4.1 \$DIST_NEXT .....	446
11.12.4.2 \$DIST_LAST .....	446
11.13 Communication .....	446
11.14 Operators .....	447
11.14.1 Arithmetic operators .....	447
11.14.2 Geometric operator .....	448
11.14.2.1 Sequence of the operands .....	449
11.14.2.2 Example of a double operation .....	449
11.14.3 Relational operators .....	451
11.14.4 Logic operators .....	452
11.14.5 Bit operators .....	452
11.14.6 Priority of the operators .....	454
11.15 Mathematical standard functions .....	455
11.16 System functions .....	457
11.16.1 DELETE_BACKWARD_BUFFER() .....	457
11.16.2 FORWARD() .....	458
11.16.3 INV_POS() .....	459
11.16.4 INVERSE() .....	459
11.16.5 ROB_STOP() and ROB_STOP_RELEASE() .....	461
11.16.6 SET_BRAKE_DELAY() .....	462

11.16.7 VARSTATE() .....	465
11.17 Editing string variables .....	467
11.17.1 Converting a string variable to a different data type .....	467
11.17.2 String variable length in the declaration .....	468
11.17.3 String variable length after initialization .....	469
11.17.4 Deleting the contents of a string variable .....	469
11.17.5 Extending a string variable .....	470
11.17.6 Searching a string variable .....	470
11.17.7 Comparing the contents of string variables .....	471
11.17.8 Copying a string variable .....	472
<b>12 Submit interpreter .....</b>	<b>473</b>
12.1 Function of the submit interpreter .....	473
12.2 Manually stopping or deselecting the Submit interpreter .....	474
12.3 Manually starting the Submit interpreter .....	474
12.4 Editing the program SPS.SUB .....	475
12.5 Creating a new SUB program .....	476
12.6 Programming .....	477
<b>13 Diagnosis .....</b>	<b>481</b>
13.1 Logbook .....	481
13.1.1 Displaying the logbook .....	481
13.1.2 “ <b>Log</b> ” tab .....	481
13.1.3 <b>Filter</b> tab .....	482
13.1.4 Configuring the logbook .....	483
13.2 Displaying the caller stack .....	483
13.3 Displaying interrupts .....	484
13.4 Displaying diagnostic data about the kernel system .....	485
13.5 Automatically compressing data for error analysis (KRCDiag) .....	486
<b>14 Installation .....</b>	<b>487</b>
14.1 System requirements .....	487
14.2 Installing Windows and the KUKA System Software (KSS) (from image) .....	487
14.3 Changing the computer name .....	490
14.4 Installing additional software .....	490
14.5 KSS update .....	492
14.5.1 Update from USB stick .....	492
14.5.2 Update from the network .....	493
<b>15 KUKA Service .....</b>	<b>495</b>
15.1 Requesting support .....	495
15.2 KUKA Customer Support .....	495
<b>Index .....</b>	<b>503</b>

# 1 Introduction

## 1.1 Target group

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

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



For optimal use of our products, we recommend that our customers take part in a course of training at KUKA College. Information about the training program can be found at [www.kuka.com](http://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 manipulator
- Documentation for the robot controller
- Operating and programming instructions for the System Software
- Instructions for options and accessories
- Parts catalog on storage medium

Each of these sets of instructions is a separate document.

## 1.3 Representation of warnings and notes

### Safety

These warnings are relevant to safety 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 INSTRUCTIONS** 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.

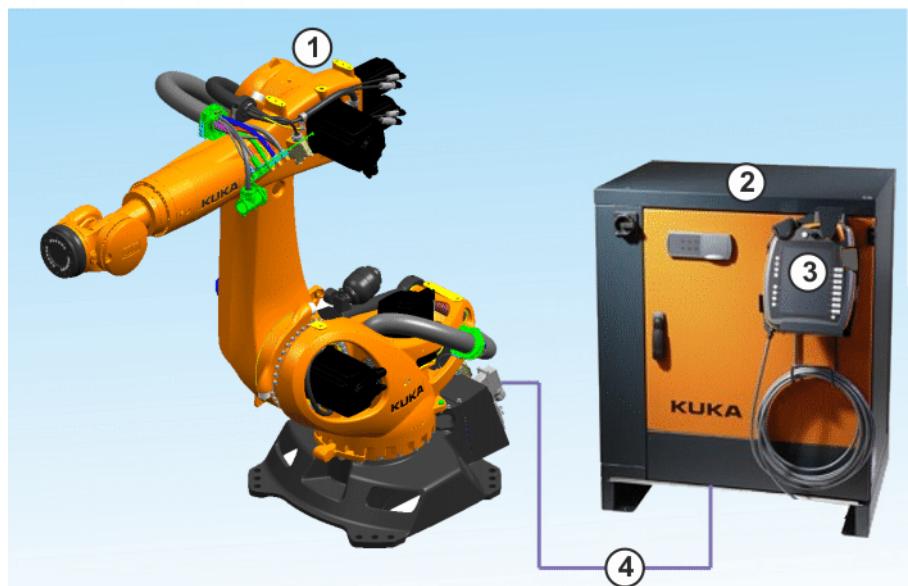
**WordPad** is a trademark of Microsoft Corporation.

## 2 Product description

### 2.1 Overview of the industrial robot

The industrial robot consists of the following components:

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



**Fig. 2-1: Example of an industrial robot**

- |                      |                       |
|----------------------|-----------------------|
| 1   Manipulator      | 3   KUKA smartPAD     |
| 2   Robot controller | 4   Connecting cables |

### 2.2 Overview of KUKA System Software (KSS)

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

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

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

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

Features:

- User administration
- Program editor
- KRL (KUKA Robot Language)

- Inline forms for programming
- Message display
- Configuration window
- etc.

(>>> 4.2 "KUKA smartHMI user interface" Page 49)

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

## 2.3 System requirements

The System Software 8.3 can be run on the following robot controller:

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

## 2.4 Intended use of the KUKA System Software

### Use

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

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

### Misuse

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

Examples of such misuse include:

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

## 2.5 KUKA USB sticks

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

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



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

### NOTICE

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

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

## 3 Safety

### 3.1 General

#### 3.1.1 Liability

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

Components of the industrial robot:

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

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

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

#### Safety information

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

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

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

#### 3.1.2 Intended use of the industrial robot

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

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

Operation of the industrial robot in accordance with its intended use also requires compliance with the operating and assembly instructions for the individual components, with particular reference to the maintenance specifications.

#### Misuse

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

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

### 3.1.3 EC declaration of conformity and declaration of incorporation

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

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

#### **EC declaration of conformity**

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

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

#### **Declaration of incorporation**

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

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

### 3.1.4 Terms used

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

Term	Description
Axis range	Range of each axis, in degrees or millimeters, within which it may move. The axis range must be defined for each axis.
Stopping distance	Stopping distance = reaction distance + braking distance The stopping distance is part of the danger zone.
Workspace	Area within which the robot may move. The workspace is derived from the individual axis ranges.
User	The user of the industrial robot can be the management, employer or delegated person responsible for use of the industrial robot.
Danger zone	The danger zone consists of the workspace and the stopping distances of the manipulator and external axes (optional).

Term	Description
Service life	<p>The service life of a safety-relevant component begins at the time of delivery of the component to the customer.</p> <p>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.</p>
KUKA smartPAD	see "smartPAD"
Manipulator	The robot arm and the associated electrical installations
Safety zone	The safety zone is situated outside the danger zone.
Safe operational stop	<p>The safe operational stop is a standstill monitoring function. It does not stop the robot motion, but monitors whether the robot axes are stationary. If these are moved during the safe operational stop, a safety stop STOP 0 is triggered.</p> <p>The safe operational stop can also be triggered externally.</p> <p>When a safe operational stop is triggered, the robot controller sets an output to the field bus. The output is set even if not all the axes were stationary at the time of triggering, thereby causing a safety stop STOP 0 to be triggered.</p>
Safety STOP 0	<p>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.</p> <p><b>Note:</b> This stop is called safety STOP 0 in this document.</p>
Safety STOP 1	<p>A stop that is triggered and monitored by the safety controller. The braking operation is carried out by the non-safety-oriented section of the robot controller and monitored by the safety controller. As soon as the manipulator has stopped, the safety controller deactivates the drives and the power supply of the brakes.</p> <p>When a safety STOP 1 is triggered, the robot controller sets an output to the field bus.</p> <p>The safety STOP 1 can also be triggered externally.</p> <p><b>Note:</b> This stop is called safety STOP 1 in this document.</p>
Safety STOP 2	<p>A stop that is triggered and monitored by the safety controller. The braking operation is carried out by the non-safety-oriented section of the robot controller and monitored by the safety controller. The drives remain activated and the brakes released. As soon as the manipulator is at a standstill, a safe operational stop is triggered.</p> <p>When a safety STOP 2 is triggered, the robot controller sets an output to the field bus.</p> <p>The safety STOP 2 can also be triggered externally.</p> <p><b>Note:</b> This stop is called safety STOP 2 in this document.</p>
Safety options	<p>Generic term for options which make it possible to configure additional safe monitoring functions in addition to the standard safety functions.</p> <p>Example: SafeOperation</p>
smartPAD	<p>Programming device for the robot controller</p> <p>The smartPAD has all the operator control and display functions required for operating and programming the industrial robot.</p>
Stop category 0	<p>The drives are deactivated immediately and the brakes are applied. The manipulator and any external axes (optional) perform path-oriented braking.</p> <p><b>Note:</b> This stop category is called STOP 0 in this document.</p>

Term	Description
Stop category 1	<p>The manipulator and any external axes (optional) perform path-maintaining braking.</p> <ul style="list-style-type: none"> <li>■ Operating mode T1: The drives are deactivated as soon as the robot has stopped, but no later than after 680 ms.</li> <li>■ Operating modes T2, AUT (KR C4), AUT EXT (KR C4), EXT (VKR C4): The drives are switched off after 1.5 s.</li> </ul> <p><b>Note:</b> This stop category is called STOP 1 in this document.</p>
Stop category 1 - Drive Ramp Stop	<p>The manipulator and any external axes (optional) perform path-oriented braking.</p> <ul style="list-style-type: none"> <li>■ Operating mode T1: The drives are deactivated as soon as the robot has stopped, but no later than after 680 ms.</li> <li>■ Operating modes T2, AUT (KR C4), AUT EXT (KR C4), EXT (VKR C4): The drives are switched off after 1.5 s.</li> </ul> <p><b>Note:</b> This stop category is called STOP 1 - DRS in this document.</p>
Stop category 2	<p>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.</p> <p><b>Note:</b> This stop category is called STOP 2 in this document.</p>
System integrator (plant integrator)	<p>The system integrator is responsible for safely integrating the industrial robot into a complete system and commissioning it.</p>
T1	Test mode, Manual Reduced Velocity (<= 250 mm/s)
T2	Test mode, Manual High Velocity (> 250 mm/s permissible)
External axis	<p>Axis of motion that does not belong to the manipulator, yet is controlled with the robot controller. e.g. KUKA linear unit, turn-tilt table, Posiflex</p>

### 3.2 Personnel

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

- User
- Personnel



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

#### User

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

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

#### Personnel

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

Personnel includes:

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

- Cleaning personnel



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

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

The system integrator is responsible for the following tasks:

- Installing the industrial robot
- Connecting the industrial robot
- Performing risk assessment
- Implementing the required safety functions and safeguards
- Issuing the EC declaration of conformity
- Attaching the CE mark
- Creating the operating instructions for the system

**Operators** The operator must meet the following preconditions:

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



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

### 3.3 Workspace, safety zone and danger zone

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

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

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

#### 3.3.1 Determining stopping distances

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

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

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

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

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

### 3.4 Triggers for stop reactions

Stop reactions of the industrial robot are triggered in response to operator actions or as a reaction to monitoring functions and error messages. The following table shows the different stop reactions according to the operating mode that has been set.

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

### 3.5 Safety functions

#### 3.5.1 Overview of the safety functions

The following safety functions are present in the industrial robot:

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

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

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

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

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

The following components are involved in the safety functions:

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

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



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



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

### 3.5.2 Safety controller

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

Safety controller tasks:

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

### 3.5.3 Selecting the operating mode

**Operating modes** The industrial robot can be operated in the following modes:

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



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

Operat-ing mode	Use	Velocities
T1	For test operation, pro-gramming and teach-ing	<ul style="list-style-type: none"> <li>■ Program verification: Programmed velocity, maxi-mum 250 mm/s</li> <li>■ Jog mode: Jog velocity, maximum 250 mm/s</li> </ul>
T2	For test operation	<ul style="list-style-type: none"> <li>■ Program verification: Programmed velocity</li> <li>■ Jog mode: Not possible</li> </ul>
AUT	For industrial robots without higher-level controllers	<ul style="list-style-type: none"> <li>■ Program operation: Programmed velocity</li> <li>■ Jog mode: Not possible</li> </ul>
AUT EXT	For industrial robots with higher-level controllers, e.g. PLC	<ul style="list-style-type: none"> <li>■ Program operation: Programmed velocity</li> <li>■ Jog mode: Not possible</li> </ul>

#### 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



If the smartPAD is fitted with a switch without a key:  
An additional device must be present to ensure that the relevant functions cannot be executed by all users, but only by a restricted group of people.

The device itself must not trigger motions of the industrial robot or other hazards. If this device is missing, death or severe injuries may result.

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

#### 3.5.4 “Operator safety” signal

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

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

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

### 3.5.5 EMERGENCY STOP device

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

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

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

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

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

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

(>>> 3.5.7 "External EMERGENCY STOP device" Page 28)

### 3.5.6 Logging off from the higher-level safety controller

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

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

Effect of the interruption:

- If a discrete safety interface is used, this triggers an EMERGENCY STOP for the overall system.

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



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



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

### 3.5.7 External EMERGENCY STOP device

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

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

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

### 3.5.8 Enabling device

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

There are 3 enabling switches installed on the smartPAD. The enabling switches have 3 positions:

- Not pressed
- Center position
- Panic position

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

- Releasing the enabling switch triggers a safety stop 2.
- Pressing the enabling switch down fully (panic position) triggers a safety stop 1.
- It is possible to hold 2 enabling switches in the center position simultaneously for up to 15 seconds. This makes it possible to adjust grip from one enabling switch to another one. If 2 enabling switches are held simultaneously in the center position for longer than 15 seconds, this triggers a safety stop 1.

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

- Press the enabling switch down fully.

- Actuate the EMERGENCY STOP device.
- Release the Start key.



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

### 3.5.9 External enabling device

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

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



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.

### 3.5.10 External safe operational stop

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

### 3.5.11 External safety stop 1 and external safety stop 2

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

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

### 3.5.12 Velocity monitoring in T1

The velocity at the mounting flange is monitored in T1 mode. If the velocity exceeds 250 mm/s, a safety stop 0 is triggered.

## 3.6 Additional protective equipment

### 3.6.1 Jog mode

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

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

### 3.6.2 Software limit switches

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

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



Further information is contained in the operating and programming instructions.

### 3.6.3 Mechanical end stops

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

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



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

### 3.6.4 Mechanical axis limitation (optional)

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

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

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



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

### 3.6.5 Options for moving the manipulator without drive energy



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

#### Description

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

- Release device (optional)

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

- Brake release device (option)  
The brake release device is designed for robot variants whose motors are not freely accessible.
- Moving the wrist axes directly by hand  
There is no release device available for the wrist axes of variants in the low payload category. This is not necessary because the wrist axes can be moved directly by hand.



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

### **NOTICE**

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

#### **3.6.6 Labeling on the industrial robot**

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

Labeling on the industrial robot consists of:

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



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

#### **3.6.7 External safeguards**

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

Physical safeguards must meet the following requirements:

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

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

- They are reduced to an absolute minimum.

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



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

## Other safety equipment

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

## 3.7 Overview of operating modes and safety functions

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

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

## 3.8 Safety measures

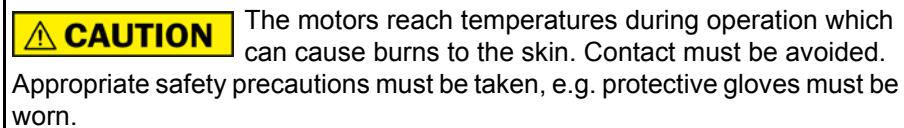
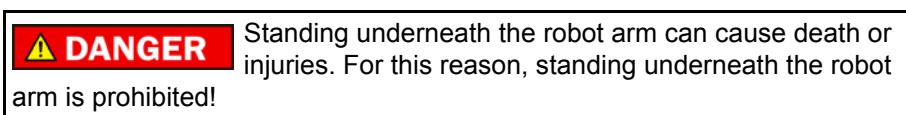
### 3.8.1 General safety measures

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

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

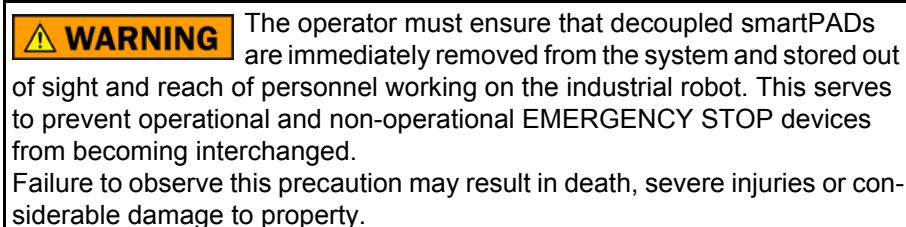


In the absence of operational safety functions and safeguards, the industrial robot can cause personal injury or material damage. If safety functions or safeguards are dismantled or deactivated, the industrial robot may not be operated.



**smartPAD** The user must ensure that the industrial robot is only operated with the smart-PAD 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.



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

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

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

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

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

### 3.8.2 Transportation

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

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

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

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

**External axis  
(optional)**

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

### 3.8.3 Start-up and recommissioning

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

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



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



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



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



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

**Function test**

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

**General test:**

It must be ensured that:

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



**WARNING** In the case of such damage, the affected components must be exchanged. In particular, the motor and counter-balancing system must be checked carefully.

External forces can cause non-visible damage. For example, it can lead to a gradual loss of drive power from the motor, resulting in unintended movements of the manipulator. Death, injuries or considerable damage to property may otherwise result.

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

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

#### **Test of the safety functions:**

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

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

#### **3.8.3.1 Checking machine data and safety configuration**



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

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

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

#### **For 6-axis robots:**

The following methods are available for performing the practical test:

- TCP calibration with the XYZ 4-point method  
The practical test is passed if the TCP has been successfully calibrated.

Or:

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

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

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

#### For palletizing robots:

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

First part:

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

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

Second part:

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

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

#### Practical test for axes that are not mathematically coupled

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

1. Mark the starting position of the axis that is not mathematically coupled.  
Also read and note the start position from the **Actual position** display on the smartHMI.
2. Move the axis manually by a freely selected path length. Determine the path length from the **Actual position** display.
  - Move linear axes a specific distance.
  - Move rotational axes through a specific angle.
3. Measure the length of the path covered and compare it with the value displayed on the smartHMI.  
The practical test is passed if the values differ by no more than 5%.
4. Repeat the test for each axis that is not mathematically coupled.

#### Practical test for robot on KUKA linear unit

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

- Move the KL manually in Cartesian mode.

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

<b>Practical test for coupleable axes</b>	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 coupleable axis. 2. Move all the remaining axes individually.  The practical test is passed if it has been possible to move all the remaining axes.
-------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

### 3.8.3.2 Start-up mode

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

The safety interface used affects "Start-up" mode:

#### Discrete safety interface

- System Software 8.2 or earlier:  
Start-up mode is always possible if all input signals at the discrete safety interface have the state "logic zero". If this is not the case, the robot controller prevents or terminates Start-up mode.  
If an additional discrete safety interface for safety options is used, the inputs there must also have the state "logic zero".
- System Software 8.3 or higher:  
Start-up mode is always possible. This also means that it is independent of the state of the inputs at the discrete safety interface.  
If an additional discrete safety interface is used for safety options: The states of these inputs are also irrelevant.

#### Ethernet safety interface

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

<b>Effect</b>	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.
---------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

#### NOTICE

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

<b>Hazards</b>	Possible hazards and risks involved in using Start-up mode:  <ul style="list-style-type: none"> <li>■ A person walks into the manipulator's danger zone.</li> <li>■ In a hazardous situation, a disabled external EMERGENCY STOP device is actuated and the manipulator is not shut down.</li> </ul> Additional measures for avoiding risks in Start-up mode:  <ul style="list-style-type: none"> <li>■ Cover disabled EMERGENCY STOP devices or attach a warning sign indicating that the EMERGENCY STOP device is out of operation.</li> <li>■ 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.</li> </ul>
----------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Use**

Intended use of Start-up mode:

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



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

**Misuse**

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

**3.8.4 Manual mode****General**

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

- Jog mode
- Teaching
- Programming
- Program verification

The following must be taken into consideration in manual mode:

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

**Setup work in T1**

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

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

tained between persons and the manipulator (including tool). Guide value: 50 cm.

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

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

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

### 3.8.5 Simulation

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

### 3.8.6 Automatic mode

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

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

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

### 3.8.7 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.



Before work is commenced on live parts of the robot system, the main switch must be turned off and secured against being switched on again. The system must then be checked to ensure that it is deenergized.

It is not sufficient, before commencing work on live parts, to execute an EMERGENCY STOP or a safety stop, or to switch off the drives, as this does not disconnect the robot system from the mains power supply. Parts remain energized. Death or severe injuries may result.

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

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

#### Robot controller

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

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

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

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

#### Counterbalancing system

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

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

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

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

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

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

#### Hazardous substances

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

- Avoid prolonged and repeated intensive contact with the skin.

- Avoid breathing in oil spray or vapors.
- Clean skin and apply skin cream.



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

### 3.8.8 Decommissioning, storage and disposal

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

### 3.8.9 Safety measures for “single point of control”

#### Overview

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

The relevant components are:

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



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

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

#### T1, T2

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

Signal	State required for SPOC
\$USER_SAF	TRUE
\$SPOC_MOTION_ENABLE	TRUE

#### Submit interpreter, PLC

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

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

Safety measures:

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

If modifications are nonetheless required, all safety-relevant signals and variables must be linked in such a way that they cannot be set to a dan-

gerous state by the submit interpreter or PLC. This is the responsibility of the system integrator.

<b>OPC server, remote control tools</b>	<p>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.</p> <p>Safety measure:</p> <p>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.</p>
<b>Tools for configu- ration of bus systems</b>	<p>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.</p> <ul style="list-style-type: none"> <li>■ WorkVisual from KUKA</li> <li>■ Tools from other manufacturers</li> </ul> <p>Safety measure:</p> <p>In the test modes, programs, outputs or other parameters of the robot controller must not be modified using these components.</p>

### 3.9 Applied norms and regulations

Name/Edition	Definition
<b>2006/42/EU:2006</b>	<p><b>Machinery Directive:</b></p> <p>Directive 2006/42/EC of the European Parliament and of the Council of 17 May 2006 on machinery, and amending Directive 95/16/EC (recast)</p>
<b>2014/30/EU:2014</b>	<p><b>EMC Directive:</b></p> <p>Directive 2014/30/EC of the European Parliament and of the Council dated 26 February 2014 on the approximation of the laws of the Member States concerning electromagnetic compatibility</p>
<b>2014/68/EU:2014</b>	<p><b>Pressure Equipment Directive:</b></p> <p>Directive 2014/68/EU of the European Parliament and of the Council dated 15 May 2014 on the approximation of the laws of the Member States concerning pressure equipment</p> <p>(Only applicable for robots with hydropneumatic counterbalancing system.)</p>
<b>EN ISO 13850:2015</b>	<p><b>Safety of machinery:</b></p> <p>Emergency stop - Principles for design</p>
<b>EN ISO 13849-1:2015</b>	<p><b>Safety of machinery:</b></p> <p>Safety-related parts of control systems - Part 1: General principles of design</p>
<b>EN ISO 13849-2:2012</b>	<p><b>Safety of machinery:</b></p> <p>Safety-related parts of control systems - Part 2: Validation</p>

<b>EN ISO 12100:2010</b>	<b>Safety of machinery:</b> General principles of design, risk assessment and risk reduction
<b>EN ISO 10218-1:2011</b>	<b>Industrial robots – Safety requirements:</b> Part 1: Robots <b>Note:</b> Content equivalent to <b>ANSI/RIA R.15.06-2012, Part 1</b>
<b>EN 614-1:2006+A1:2009</b>	<b>Safety of machinery:</b> Ergonomic design principles - Part 1: Terms and general principles
<b>EN 61000-6-2:2005</b>	<b>Electromagnetic compatibility (EMC):</b> Part 6-2: Generic standards; Immunity for industrial environments
<b>EN 61000-6-4:2007 + A1:2011</b>	<b>Electromagnetic compatibility (EMC):</b> Part 6-4: Generic standards; Emission standard for industrial environments
<b>EN 60204-1:2006/A1:2009</b>	<b>Safety of machinery:</b> Electrical equipment of machines - Part 1: General requirements



## 4 Operation

### 4.1 KUKA smartPAD teach pendant

#### 4.1.1 Front view

##### Function

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.

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

##### Overview

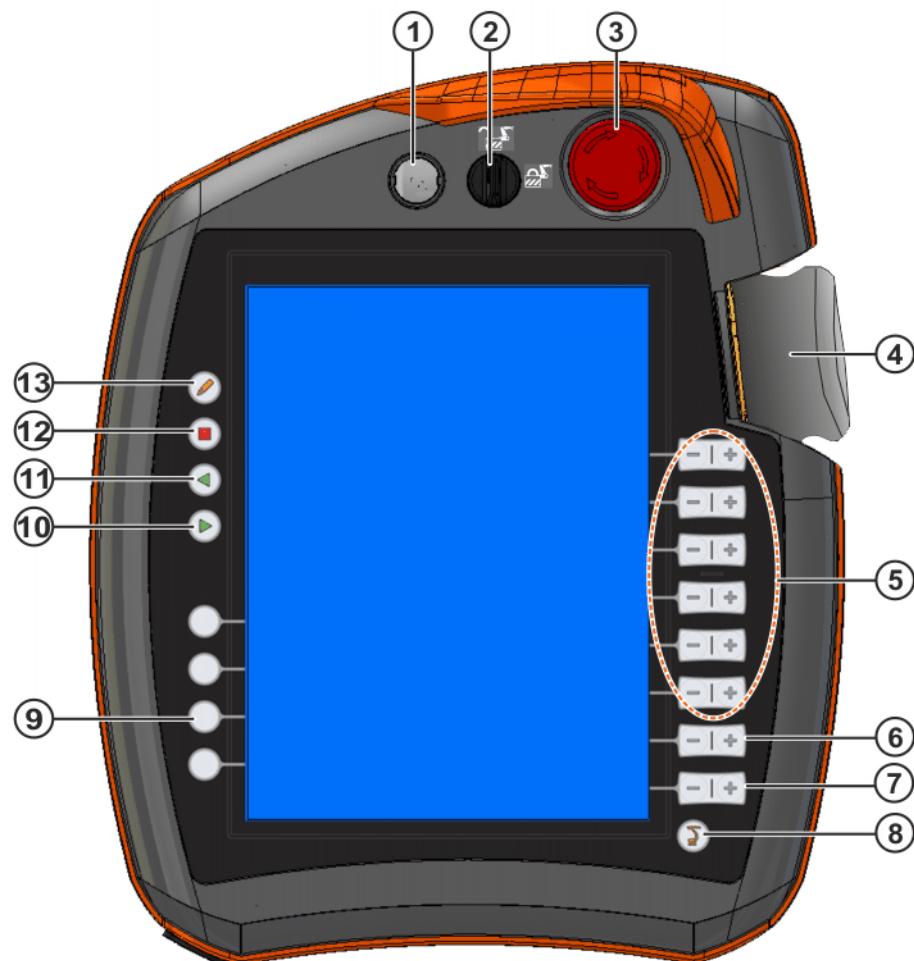


Fig. 4-1: KUKA smartPAD, front view

Item	Description
1	Button for disconnecting the smartPAD  (>>> 4.1.3 "Disconnecting and connecting the smartPAD" Page 48)
2	Mode selector switch. The switch may be one of the following variants: <ul style="list-style-type: none"><li>■ With key</li><li>■ Without key</li></ul> The mode selector switch is used to call the connection manager. The operating mode can be changed by using the connection manager.  (>>> 4.13 "Changing operating mode" Page 63)
3	EMERGENCY STOP device Stops the robot in hazardous situations. The EMERGENCY STOP button locks itself in place when it is pressed.
4	Space Mouse: For moving the robot manually
5	Jog keys: For moving the robot manually
6	Key for setting the program override
7	Key for setting the jog override
8	Main menu key: Shows the menu items on the smartHMI. It can also be used for creating screenshots.
9	Status keys. The status keys are used primarily for setting parameters in technology packages. Their exact function depends on the technology packages installed.
10	Start key: The Start key is used to start a program.
11	Start backwards key: The Start backwards key is used to start a program backwards. The program is executed step by step.
12	STOP key: The STOP key is used to stop a program that is running.
13	Keyboard key  Displays the keyboard. It is generally not necessary to press this key to display the keyboard, as the smartHMI detects when keyboard input is required and displays the keyboard automatically.  (>>> 4.2.1 "Keypad" Page 50)

#### 4.1.2 Rear view

##### Overview



**Fig. 4-2: KUKA smartPAD, rear view**

- |   |                   |   |                      |
|---|-------------------|---|----------------------|
| 1 | Enabling switch   | 4 | USB connection       |
| 2 | Start key (green) | 5 | Enabling switch      |
| 3 | Enabling switch   | 6 | Identification plate |

Description	Element	Description
	<b>Rating plate</b>	Rating plate
	<b>Start key</b>	The Start key is used to start a program.
	<b>Enabling switch</b>	<p>The enabling switch has 3 positions:</p> <ul style="list-style-type: none"> <li>■ Not pressed</li> <li>■ Center position</li> <li>■ Fully pressed (panic position)</li> </ul> <p>The enabling switch must be held in the center position in operating modes T1 and T2 in order to be able to jog the manipulator.</p> <p>In the operating modes Automatic and Automatic External, the enabling switch has no function.</p>
	<b>USB connection</b>	<p>The USB connection is used, for example, for archiving and restoring data.</p> <p>Only for FAT32-formatted USB sticks.</p>

### 4.1.3 Disconnecting and connecting the smartPAD

#### Description

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



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

#### Procedure

##### Disconnection:

1. Press the disconnect button on the smartPAD.

A message and a counter are displayed on the smartHMI. The counter runs for 30 s. During this time, the smartPAD can be disconnected 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.

2. Disconnect the smartPAD 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.

##### Connection:

- Connect the smartPAD to the robot controller.

A smartPAD can be connected at any time. Precondition: Same smartPAD variant as the disconnected device. 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.



The user connecting a smartPAD to the robot controller must subsequently stay with the smartPAD for at least 30 s, i.e. until the EMERGENCY STOP and enabling switches are operational once again. This prevents another user from trying to activate a non-operational EMERGENCY STOP in an emergency situation, for example. Failure to observe this can lead to death, injury or property damage.

## 4.2 KUKA smartHMI user interface

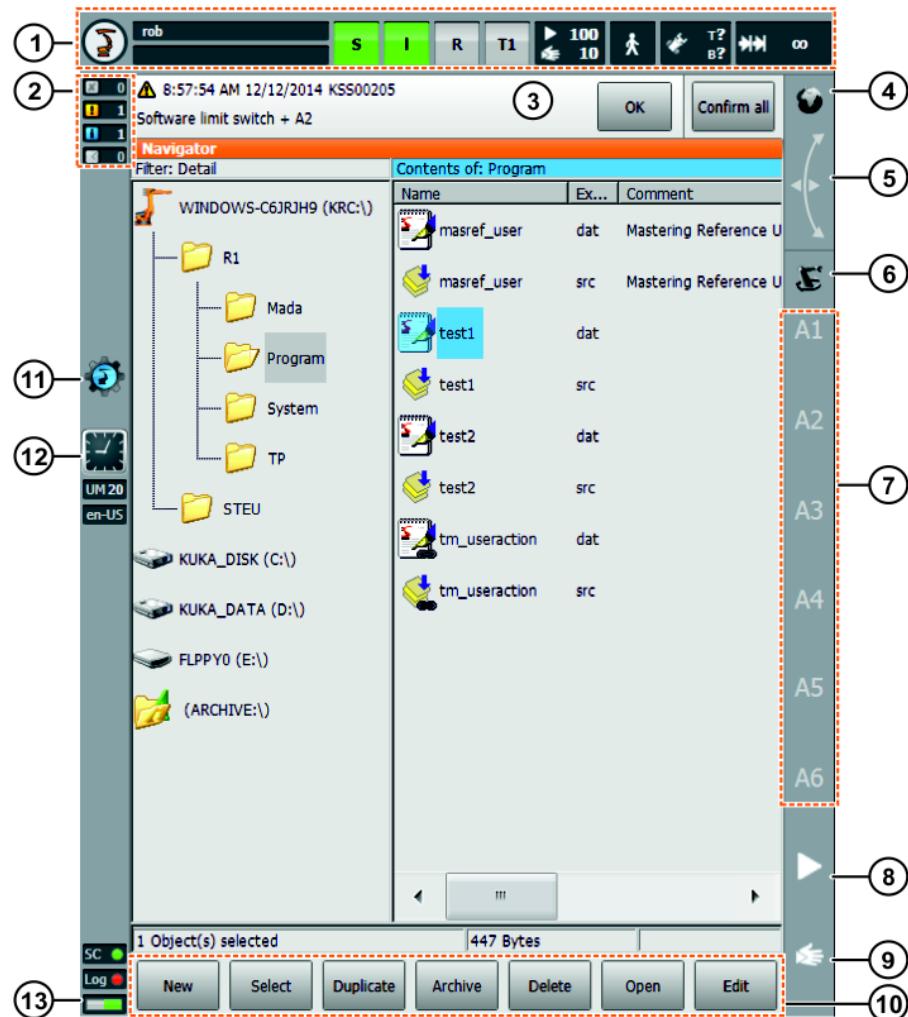


Fig. 4-3: KUKA smartHMI user interface

Item	Description
1	Status bar ( <b>&gt;&gt;&gt; 4.2.2 "Status bar" Page 51</b> )
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 <b>OK</b> . All acknowledgeable messages can be acknowledged at once with <b>All OK</b> .
4	Space Mouse status indicator This indicator shows the current coordinate system for jogging with the Space Mouse. Touching the indicator displays all coordinate systems, allowing a different one to be selected.

Item	Description
5	<b>Space Mouse alignment</b> indicator Touching this indicator opens a window in which the current alignment of the Space Mouse is indicated and can be changed. (>>> 4.15.8 "Defining the alignment of the Space Mouse" Page 74)
6	<b>Jog keys</b> status indicator This indicator shows the current coordinate system for jogging with the jog keys. Touching the indicator displays all coordinate systems, allowing a different one to be selected.
7	Jog key labels If axis-specific jogging is selected, the axis numbers are displayed here (A1, A2, etc.). If Cartesian jogging is selected, the coordinate system axes are displayed here (X, Y, Z, A, B, C). Touching the label causes the selected kinematics group to be displayed.
8	Program override (>>> 8.5 "Setting the program override (POV)" Page 273)
9	Jog override (>>> 4.15.3 "Setting the jog override (HOV)" Page 70)
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 <b>Edit</b> button. This can be used to call numerous commands relating to the Navigator.
11	WorkVisual icon Touching the icon takes you to the <b>Project management</b> window. (>>> 7.11.3 "Project management window" Page 258)
12	Clock The clock displays the system time. Touching the clock displays the system time in digital format, together with the current date.
13	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

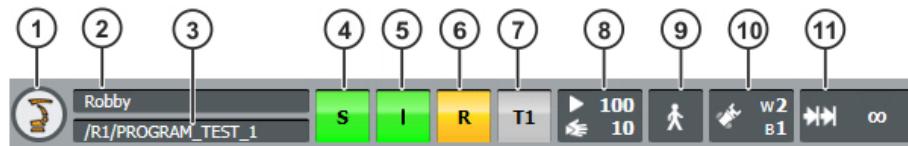


Fig. 4-5: KUKA smartHMI status bar

Item	Description
1	Main menu key. Shows the menu items on the smartHMI.
2	Robot name. The robot name can be changed.
3	If a program has been selected, the name is displayed here.
4	<b>Submit interpreter</b> status indicator
5	<b>Drives</b> 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 52)
6	<b>Robot interpreter</b> status indicator. Programs can be reset or canceled here. (>>> 8.6 "Robot interpreter status indicator" Page 273)
7	Current operating mode
8	<b>POV/HOV</b> status indicator. Indicates the current program override and the current jog override.
9	<b>Program run mode</b> status indicator. Indicates the current program run mode.
10	<b>Tool/base</b> status indicator. Indicates the current tool and base.
11	<b>Incremental jogging</b> 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	I	I	O

Meaning of the symbols and colors:

Symbol: I	The drives are ON. (\$PERI_RDY == TRUE) <ul style="list-style-type: none"> <li>■ The intermediate circuit is fully charged.</li> </ul>
Symbol: O	The drives are OFF. (\$PERI_RDY == FALSE) <ul style="list-style-type: none"> <li>■ The intermediate circuit is not charged or incompletely charged.</li> </ul>
Color: Green	\$COULD_START_MOTION == TRUE <ul style="list-style-type: none"> <li>■ The enabling switch has been pressed (center position) or is not required.</li> <li>■ And: There are no active messages preventing robot motion.</li> </ul>
Color: Gray	\$COULD_START_MOTION == FALSE <ul style="list-style-type: none"> <li>■ The enabling switch has not been pressed or fully pressed.</li> <li>■ And/or: There are active messages preventing robot motion.</li> </ul>



- 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.

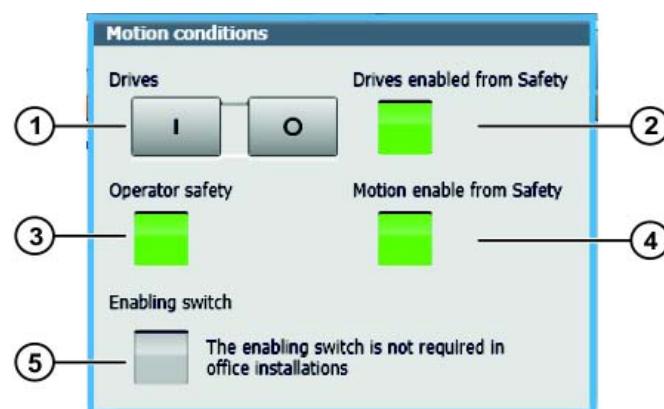


Fig. 4-6: “Motion conditions” window

Item	Description
1	I: Touch to switch on the drives. O: Touch to switch off the drives.
2	<b>Green:</b> The drives enable signal has been received from the safety controller. <b>Gray:</b> 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	<b>Operator safety</b> signal <b>Green:</b> \$USER_SAF == TRUE <b>Gray:</b> \$USER_SAF == FALSE (>>> "\$USER_SAF == TRUE" Page 53)
4	<b>Green:</b> The motion enable signal has been received from the safety controller. <b>Gray:</b> The safety controller has triggered a safety stop 1 or a safety stop 2. No motion enable present. <b>Note:</b> The status of <b>Motion enable from Safety</b> does not correlate with the status of \$MOVE_ENABLE!
5	<b>Green:</b> The enabling switch is pressed (center position). <b>Gray:</b> The enabling switch has not been pressed or fully pressed, or is not required.

#### \$USER\_SAF == TRUE

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

Control	Operating mode	Condition
KR C4	T1, T2	<ul style="list-style-type: none"> <li>■ The enabling switch is pressed.</li> </ul>
	AUT, AUT EXT	<ul style="list-style-type: none"> <li>■ The physical safeguard is closed.</li> </ul>
VKR C4	T1	<ul style="list-style-type: none"> <li>■ The enabling switch is pressed.</li> <li>■ E2/E22 is closed.</li> </ul>
	T2	<ul style="list-style-type: none"> <li>■ The enabling switch is pressed.</li> <li>■ E2/E22 and E7 are closed</li> </ul>
	EXT	<ul style="list-style-type: none"> <li>■ The physical safeguard is closed.</li> <li>■ E2/E22 and E7 are open.</li> </ul>

#### 4.2.4 Minimizing KUKA smartHMI (displaying Windows interface)

##### Precondition

- User group "Expert".
- Operating mode T1 or T2.

##### Procedure

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



## 4.3 Switching on the robot controller and starting the KSS

### Procedure

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

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

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

## 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. **Configure**).  
Depending on how many nested submenus are open, the **Main Menu** column may no longer be visible, with only the submenus remaining visible.
- The arrow key in the top right-hand corner closes the most recently opened submenu.
- The Home key in the top right-hand corner closes all open submenus.
- The most recently selected menu items are displayed in the bottom section (maximum 6).  
This makes it possible to select these menu items again directly without first having to close other submenus that might be open.
- The white cross on the left-hand side closes the window.

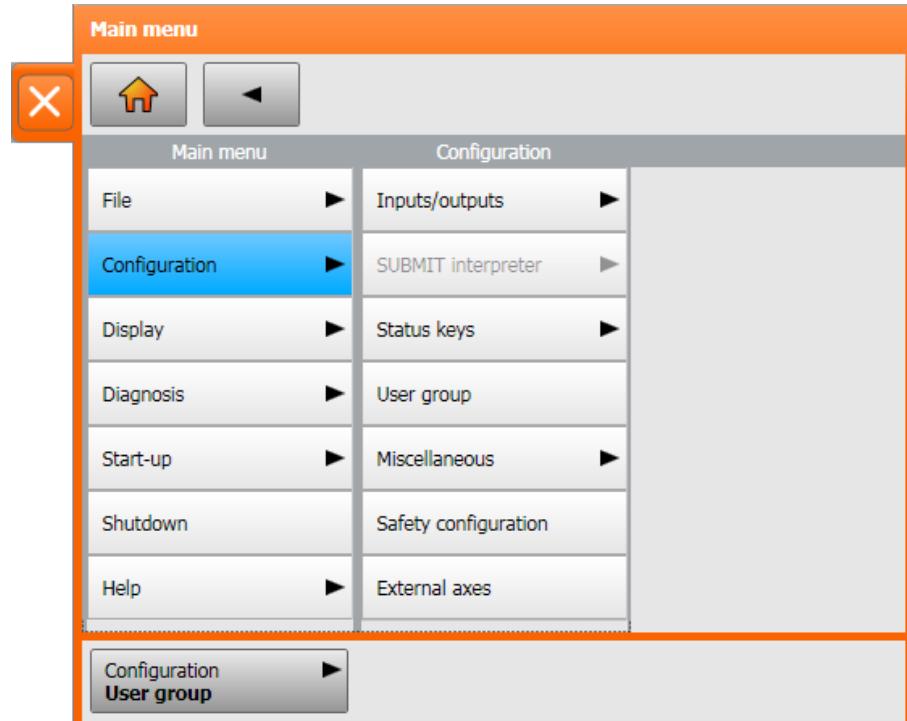


Fig. 4-7: Example: Configuration submenu is open.

## 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**

- “Expert” user group

**Procedure**

1. In the main menu, select **Shutdown**. A window opens.
2. Select the start type: **Cold start** or **Hibernate**.  
(>>> "Start types" Page 57)
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 "Exiting or restarting KSS" Page 55)

## 4.6 Exiting or restarting KSS

**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.

**Precondition**

- For certain options: “Expert” user group

### NOTICE

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

If this option was not selected on shutting down, the main switch can be pressed once the controller has shut down.

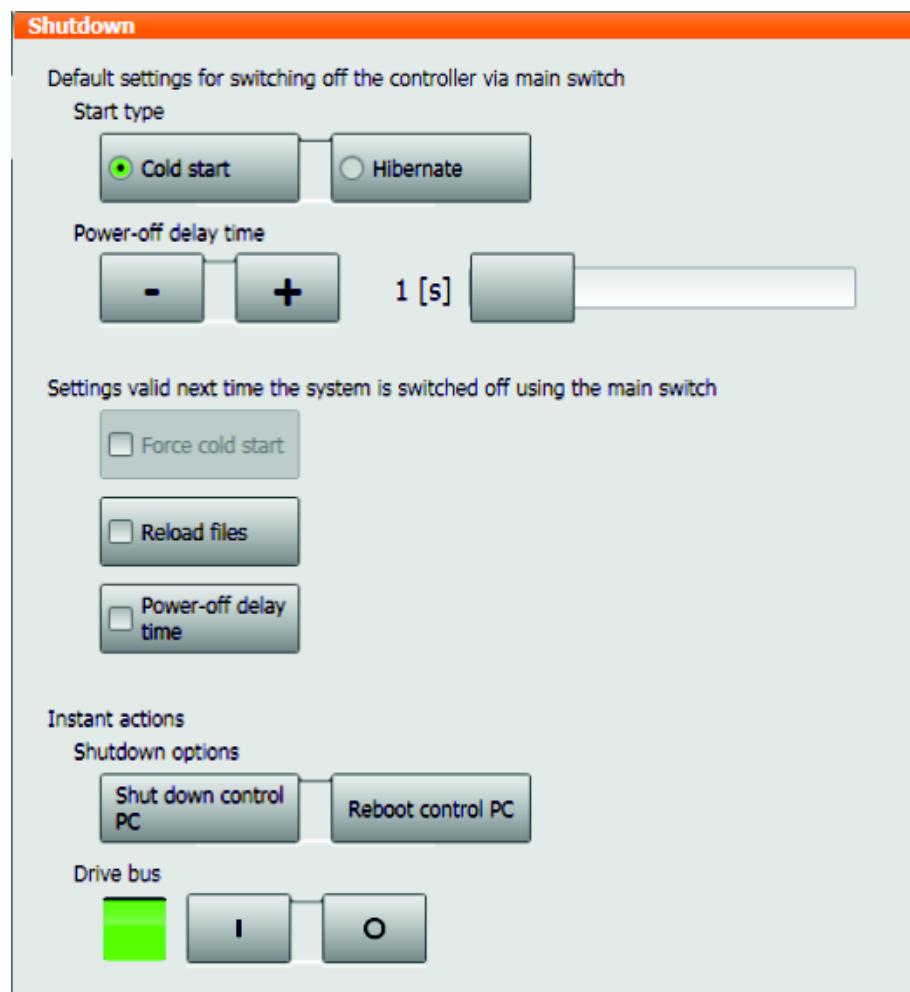
**Procedure**

1. Select the menu item **Shutdown** in the main menu.
2. Select the desired options.
3. Press **Shutdown 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
<b>Default settings for switching off the system</b>	
These settings can only be modified in the user group “Expert”.	
<b>Cold start</b>	<b>Cold start</b> is the standard start type. ( <a href="#">&gt;&gt;&gt; "Start types" Page 57</a> )
<b>Hibernate</b>	<b>Hibernate</b> is the standard start type.
<b>Power-off delay time</b>	<p>If the robot controller is switched off at the main switch, it is only shut down after the delay time defined here. During the delay time, the robot controller is powered by its battery.</p> <p>The delay time can only be modified in the user group “Expert”.</p> <p><b>Note:</b></p> <ul style="list-style-type: none"> <li>■ The power-off delay time only applies if the voltage is switched off via the main switch. The power failure delay time applies for genuine power failures.</li> <li>■ Exception for “KR C4 compact”: The power-off delay time has no function for this controller variant! The power failure delay time applies here, even when switching off via the main switch.</li> </ul> <p>(<a href="#">&gt;&gt;&gt; 4.6.1 "Shutting down after power failure" Page 58</a>)</p>
<b>Settings that are only valid next time the system is switched off</b>	
<b>Force cold start</b>	Activated: The next start is a cold start. Only available if <b>Hibernate</b> has been selected.

Option	Description
<b>Reload files</b>	<p>Activated: The next start is an initial cold start.</p> <p>This option must be selected in the following cases:</p> <ul style="list-style-type: none"> <li>■ 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.)</li> <li>■ If hardware components are to be exchanged after shutdown.</li> </ul> <p>Can only be selected in the “Expert” user group. Only available if <b>Cold start</b> or <b>Force cold start</b> has been selected.</p> <p>Depending on the hardware, the initial cold start takes approx. 30 to 150 seconds longer than a normal cold start.</p>
<b>Power-off delay time</b>	<p>Activated: The delay time is adhered to the next time the system is shut down.</p> <p>Deactivated: The delay time is ignored the next time the system is shut down.</p>
<b>Instant actions</b>	
Only available in operating modes T1 and T2.	
<b>Shut down control PC</b>	The robot controller shuts down.
<b>Reboot control PC</b>	The robot controller shuts down and then reboots with a cold start.
<b>Drive bus OFF / ON</b>	<p>The drive bus can be switched off or on.</p> <p><b>Drive bus status</b> indicator:</p> <ul style="list-style-type: none"> <li>■ Green: Drive bus is on.</li> <li>■ Red: Drive bus is off.</li> <li>■ Gray: Status of the drive bus is unknown.</li> </ul>

Start types	Start type	Description
	Cold start	<p>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.</p> <p><b>Note:</b> 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 <b>Reload files</b>. This cold start is called an “initial cold start”.</p> <p>In the case of a cold start without <b>Reload files</b>, these changes are not taken into consideration.</p>
	Hibernate	<p>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.</p> <p>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.</p>

#### 4.6.1 Shutting down after power failure

In the case of a power failure, the robot comes to a standstill. However, the robot controller does not shut down immediately but rather only after the power failure delay time. In other words, brief power failures are overridden through this delay time. The error messages must then only be acknowledged and the program can be resumed.

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

Robot controller	Power failure delay time
"KR C4 compact" variant	1 s
All other variants of KR C4	3 s

If the power failure lasts longer than the power failure delay time and the robot controller shuts down, then the standard start type defined in the **Shutdown** window applies for the restart.



- The power failure delay time does not apply if the voltage is switched off via the main switch. The power-off delay time applies for this.  
Exception for "KR C4 compact": For this controller variant, the power failure delay time also applies when switching off via the main switch.
- The power failure delay time is particularly important for systems without a reliable mains supply. Delay times of up to 240 s are possible. If the existing times are to be modified, please contact KUKA Deutschland GmbH.

#### 4.7 Switching drives on/off

##### 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 52)
2. Switch the drives on or off.

#### 4.8 Switching the robot controller off

##### Description

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

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

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

##### Procedure

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



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

## 4.9 Setting the user interface language

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

**Description** The following languages are available:

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

## 4.10 Creating a screenshot on the smartPAD

- Procedure**
- Press the main menu key in the bottom right-hand corner of the smartPAD twice.

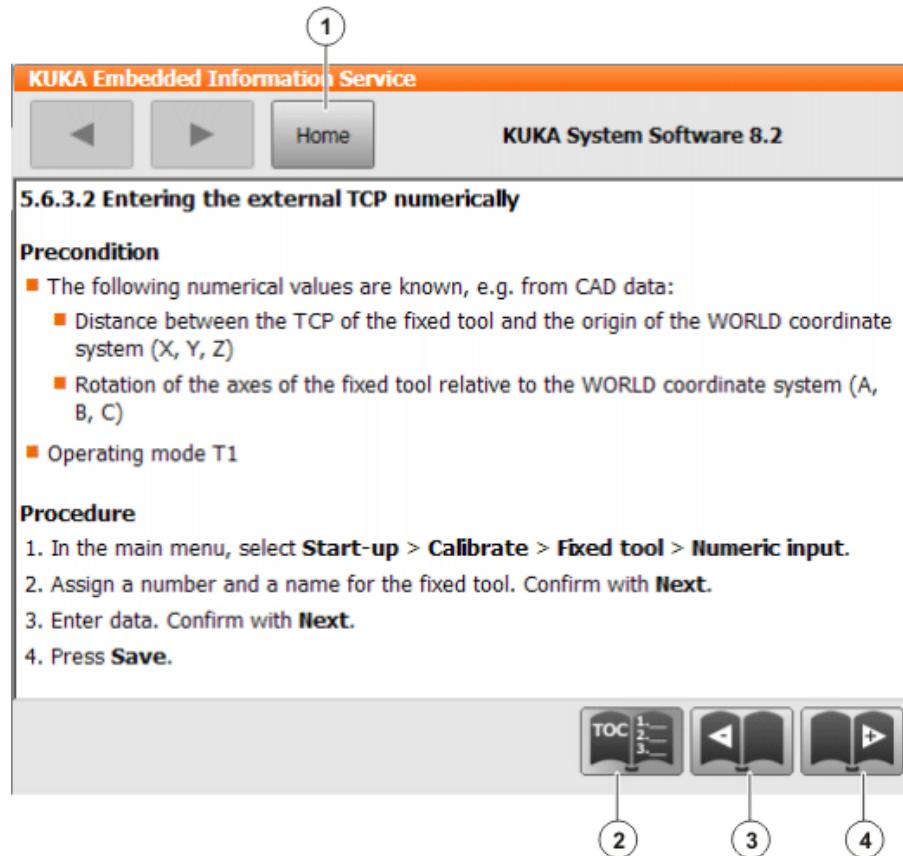
The screenshot is saved in the directory C:/KUKA/Screenshot.

## 4.11 Online documentation and help for messages

### 4.11.1 Calling online documentation

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

- Procedure**
1. In the main menu, select **Help > Documentation**. Then select either **System Software** or the menu item for the technology package.  
The **KUKA Embedded Information Service** window is opened. The table of contents of the documentation is displayed.
  2. Touch a chapter. The topics it contains are displayed.
  3. Touch a topic. The description is displayed.

**Example**

**Fig. 4-9: 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.

#### 4.11.2 Calling help for the messages

**Description**

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

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

**Procedure**
**Calling the help for a message in the message window**

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

1. Touch the question mark. The **KUKA Embedded Information Service – Message page** window is opened.  
The window contains a variety of information about the message.  
(>>> Fig. 4-10 )
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.

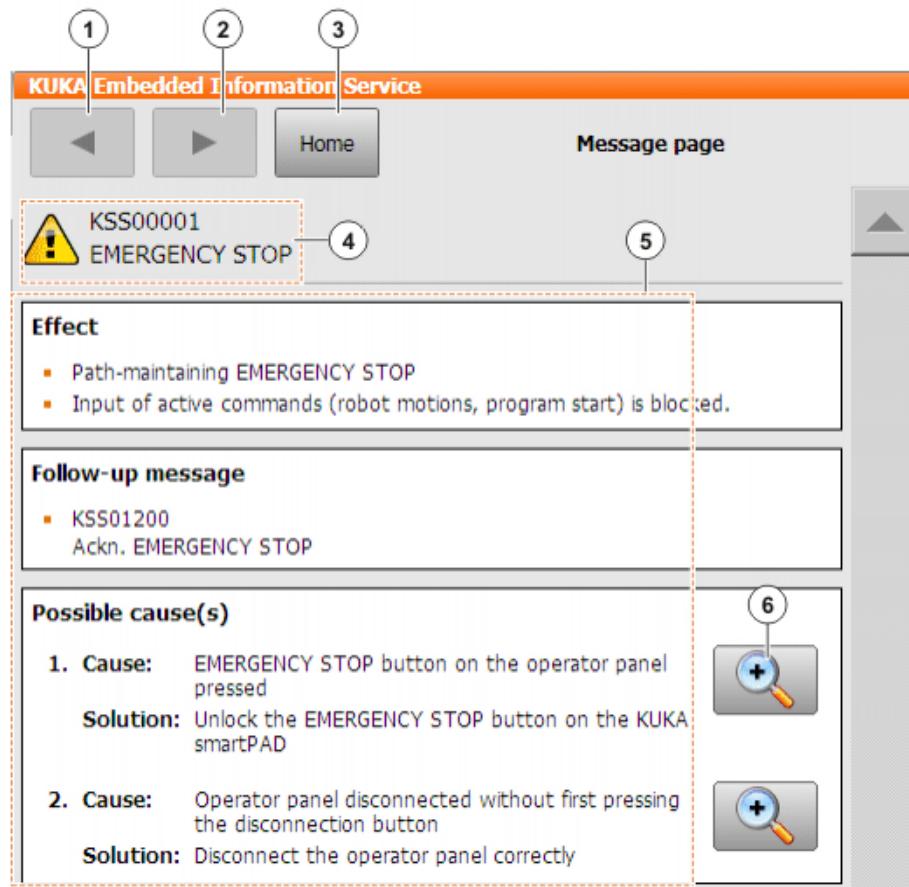
(>>> Fig. 4-10 )

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. ( <a href="#">&gt;&gt;&gt; Fig. 4-11</a> )

### Detail page

**KUKA Embedded Information Service**

Detail page

**KSS00001**  
**EMERGENCY STOP**

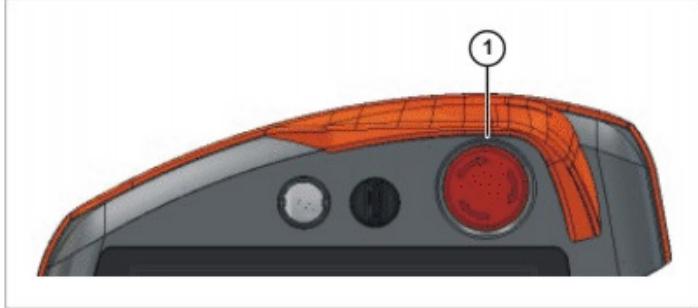
**Cause:** EMERGENCY STOP button on the operator panel pressed

**Description**  
The EMERGENCY STOP button on the KCP has been pressed.

**Solution:** Unlock the EMERGENCY STOP button on the KUKA smartPAD

**Procedure**

- Turn the EMERGENCY STOP button clockwise (direction indicated by the arrow) to release it (position 1).



EMERGENCY STOP button on the KUKA smartPAD

◀ + 🔍 ▶ 1/2

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

## 4.12 Changing user group

### Description

Different functions are available in the KSS, depending on the user group. The following user groups are available:

#### ■ Operator

User group for the operator. This is the default user group.

#### ■ User

User group for the operator. (By default, the user groups “Operator” and “User” are defined for the same target group.)

- **Expert**

User group for the programmer. This user group is protected by means of a password.

- **Safety recovery**

This user group can activate and configure the safety configuration of the robot. This user group is protected by means of a password.

- **Safety maintenance**

This user group is only relevant if a safety option is used, e.g. KUKA.SafeOperation. The user group is protected by means of a password.

- **Administrator**

The range of functions is the same as that for the user group "Expert". It is additionally possible, in this user group, to integrate plug-ins into the robot controller.

This user group is protected by means of a password.

The default password is "kuka". The password must be changed.

(>>> 6.11 "Changing the password" Page 176)

When the system is booted, the default user group is selected.

If the mode is switched to AUT or AUT EXT, the robot controller switches to the default user group for safety reasons. If a different user group is desired, this must be selected subsequently.

If no actions are carried out in the user interface within a certain period of time, the robot controller switches to the default user group for safety reasons. The default setting is 300 s.

**Procedure**

1. Select **Configuration > User group** in the main menu. The current user group is displayed.
2. Press **Default** to switch to the default user group. (**Default** is not available if the default user group is already selected.)  
Press **Login...** to switch to a different user group. Select the desired user group.
3. If prompted: Enter password and confirm with **Log-on**.

## 4.13 Changing operating mode



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

**Precondition**

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

**Procedure**

1. Turn the mode selector switch on the smartPAD. The connection manager is displayed.
2. Select the operating mode. (>>> 3.5.3 "Selecting the operating mode" Page 25)
3. Return the mode selector switch to its original position.  
The selected operating mode is displayed in the status bar of the smart-PAD.

Operat-ing mode	Use	Velocities
T1	For test operation, pro-gramming and teach-ing	<ul style="list-style-type: none"> <li>■ Program verification: Programmed velocity, maxi-mum 250 mm/s</li> <li>■ Jog mode: Jog velocity, maximum 250 mm/s</li> </ul>
T2	For test operation	<ul style="list-style-type: none"> <li>■ Program verification: Programmed velocity</li> <li>■ Jog mode: Not possible</li> </ul>
AUT	For industrial robots without higher-level controllers	<ul style="list-style-type: none"> <li>■ Program operation: Programmed velocity</li> <li>■ Jog mode: Not possible</li> </ul>
AUT EXT	For industrial robots with higher-level controllers, e.g. PLC	<ul style="list-style-type: none"> <li>■ Program operation: Programmed velocity</li> <li>■ Jog mode: Not possible</li> </ul>

## 4.14 Coordinate systems

### Overview

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

- WORLD
- ROBROOT
- BASE
- TOOL

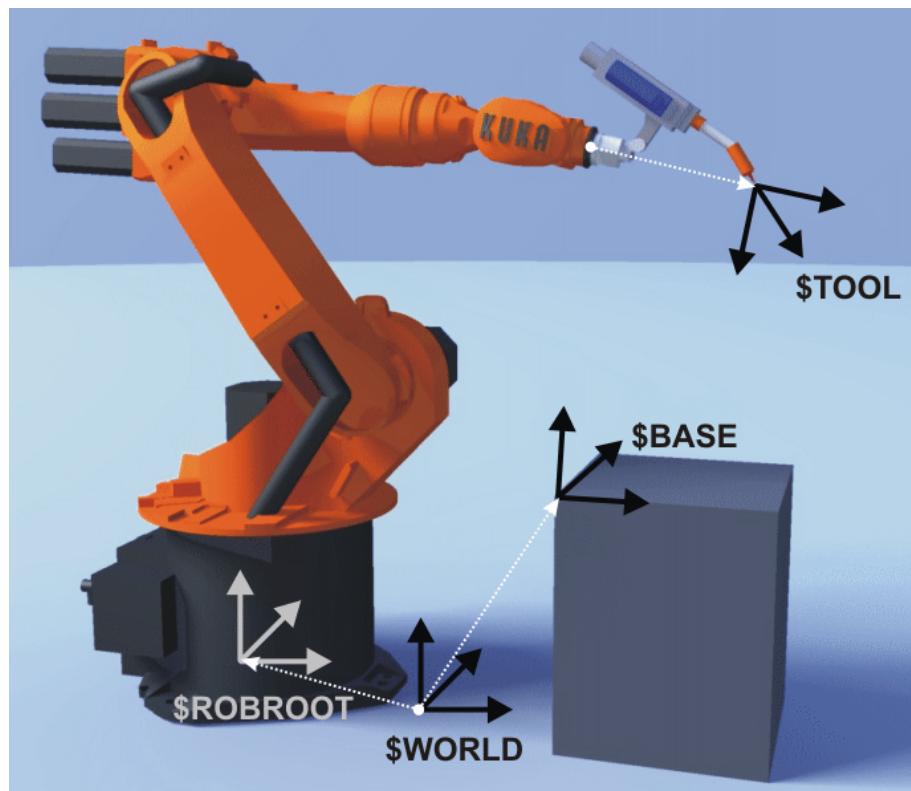


Fig. 4-12: Overview of coordinate systems

### 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.

(>>> 5.12.3 "Base calibration" Page 133)

### **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.

(>>> 5.12.2 "Tool calibration" Page 126)

### **Angles of rotation of the robot coordinate systems**

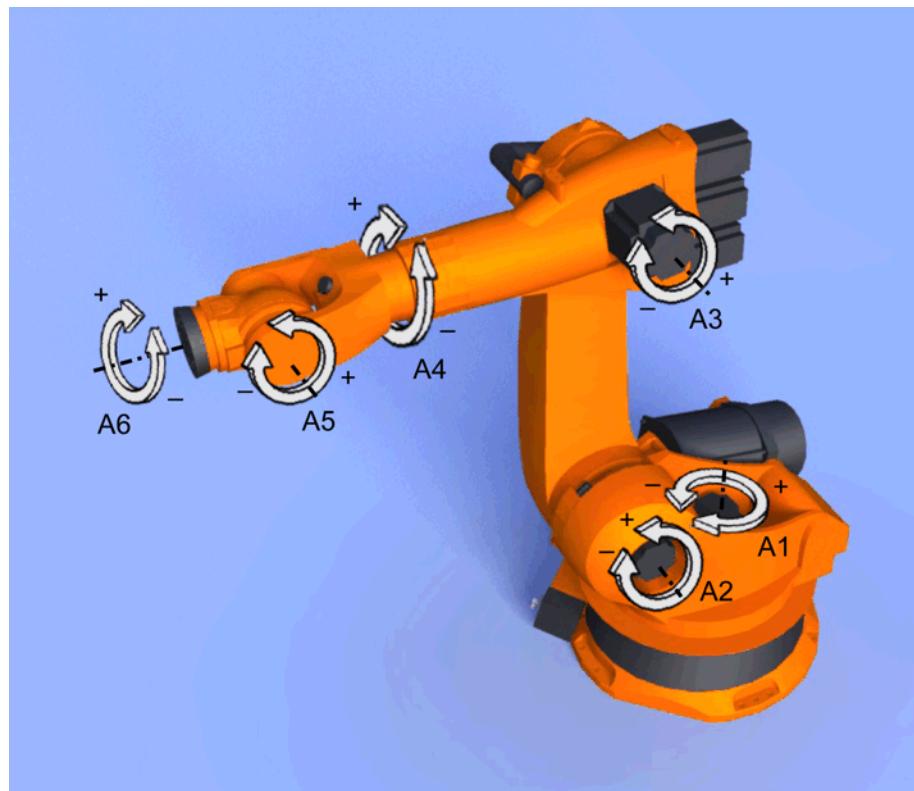
Angle	Rotation about axis
Angle A	Rotation about the Z axis
Angle B	Rotation about the Y axis
Angle C	Rotation about the X axis

## **4.15 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. 4-13: 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	(>>> 4.15.6 "Cartesian jogging with the jog keys" Page 71)	(>>> 4.15.5 "Axis-specific jogging with the jog keys" Page 71)
Space Mouse	(>>> 4.15.9 "Cartesian jogging with the Space Mouse" Page 75)	Axis-specific jogging with the Space Mouse is possible, but is not described here.

#### 4.15.1 “Jog options” window

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

**Procedure** To open the **Jogging options** window:

1. Open a status indicator on the smartHMI, e.g. the **Cur. tool/base** status indicator.  
(Not possible for the **Submit interpreter**, **Drives** and **Robot interpreter** status indicators.)  
A window opens.
2. Press **Options**. The **Jogging Options** window is opened.

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

#### 4.15.1.1 “General” tab

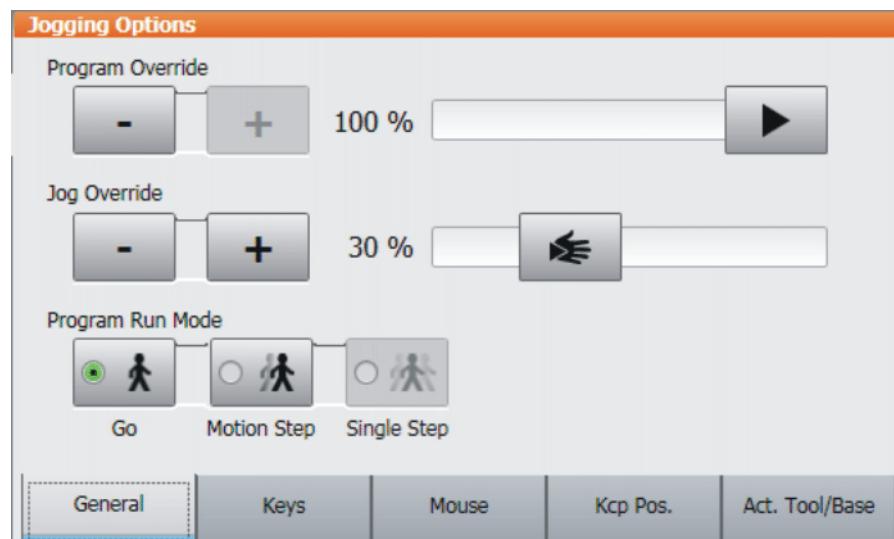


Fig. 4-14: General tab

#### Description

Item	Description
1	Set the program override. (>>> 8.5 "Setting the program override (POV)" Page 273)
2	Set the jog override. (>>> 4.15.3 "Setting the jog override (HOV)" Page 70)
3	Select the program run mode. (>>> 8.2 "Program run modes" Page 269)

#### 4.15.1.2 “Keys” tab

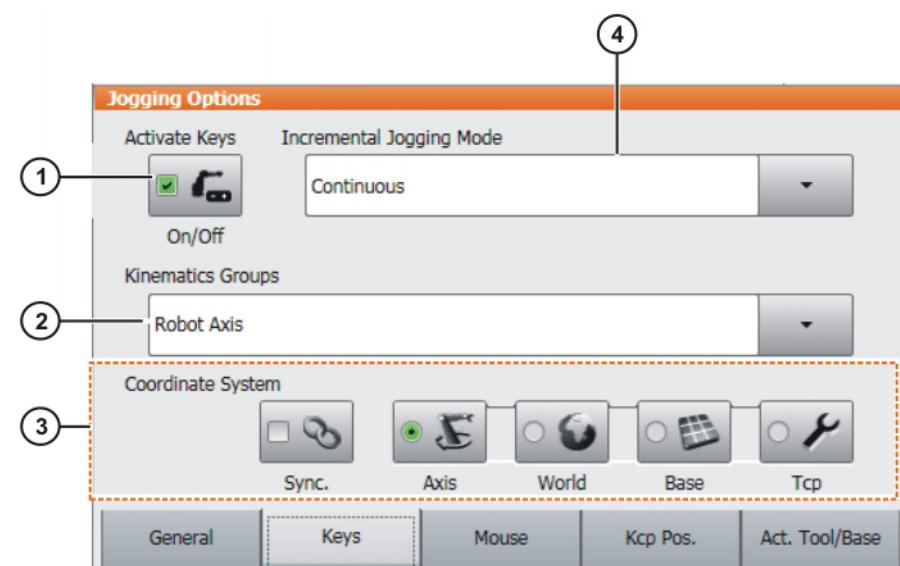


Fig. 4-15: Keys tab

Description	Item	Description
	1	Activate jog mode "Jog keys" (>>> 4.15.2 "Activating the jog mode" Page 70)
	2	Select a kinematics group. The kinematics group defines the axes to which the jog keys refer. Default: <b>Robot axes</b> (= A1 to A6) Depending on the system configuration, other kinematics groups may be available. (>>> 4.16 "Jogging external axes" Page 76)
	3	Select the coordinate system for jogging with the jog keys: <ul style="list-style-type: none"> <li>■ <b>Axes, World, Base or Tool</b></li> </ul> Check box <b>Sync.:</b> <ul style="list-style-type: none"> <li>■ Check box not active (default): On the <b>Keys</b> and <b>Mouse</b> tabs, different coordinate systems can be selected.</li> <li>■ Check box active: On the <b>Keys</b> and <b>Mouse</b> tabs, only one coordinate system can be selected, which is the same in each case. If the coordinate system is changed on one tab, the setting on the other is adapted automatically.</li> </ul>
	4	Incremental jogging (>>> 4.15.10 "Incremental jogging" Page 75)

#### 4.15.1.3 "Mouse" tab

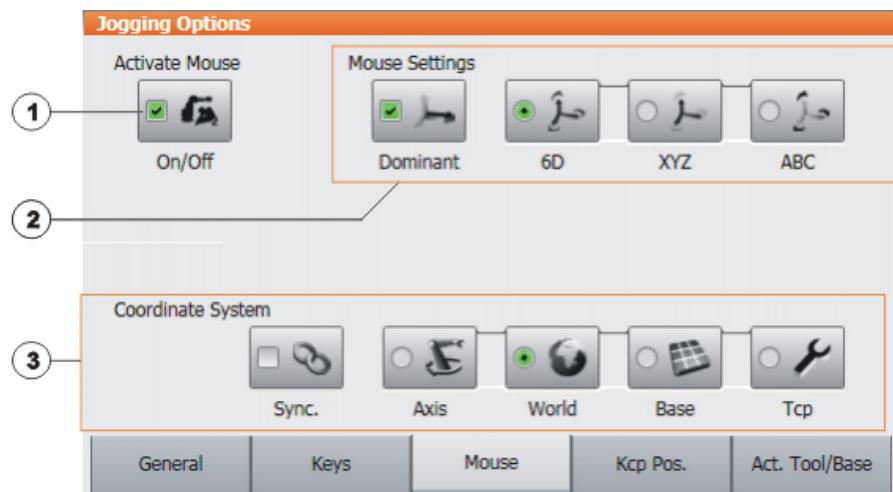


Fig. 4-16: Mouse tab

Description	Item	Description
	1	Activate jog mode "Space Mouse" (>>> 4.15.2 "Activating the jog mode" Page 70)

Item	Description
2	Configure the Space Mouse (>>> 4.15.7 "Configuring the Space Mouse" Page 72)
3	Select the coordinate system for jogging with the Space Mouse: <ul style="list-style-type: none"> <li>■ <b>Axes, World, Base or Tool</b></li> </ul> <p>Check box <b>Sync.:</b></p> <ul style="list-style-type: none"> <li>■ Check box not active (default): On the <b>Keys</b> and <b>Mouse</b> tabs, different coordinate systems can be selected.</li> <li>■ Check box active: On the <b>Keys</b> and <b>Mouse</b> tabs, only one coordinate system can be selected, which is the same in each case. If the coordinate system is changed on one tab, the setting on the other is adapted automatically.</li> </ul>

#### 4.15.1.4 "KCP pos." tab

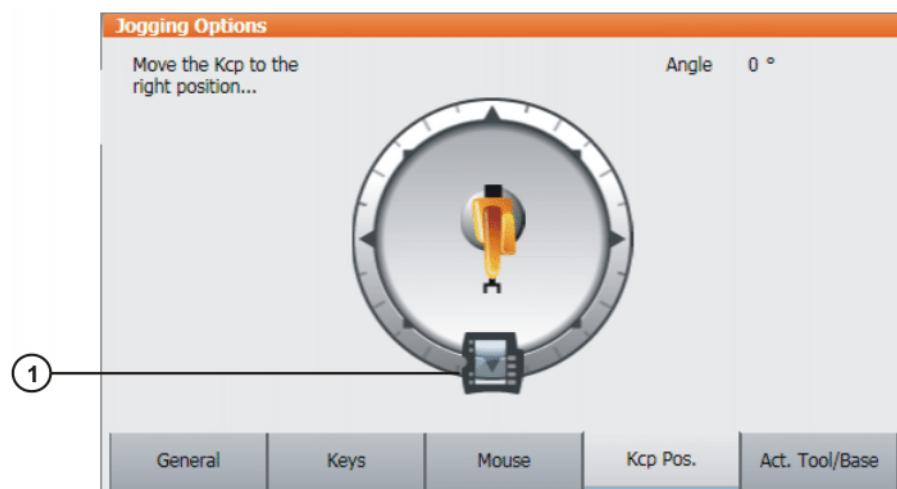


Fig. 4-17: Kcp Pos. tab

#### Description

Item	Description
1	(>>> 4.15.8 "Defining the alignment of the Space Mouse" Page 74)

#### 4.15.1.5 “Cur. tool/base” tab

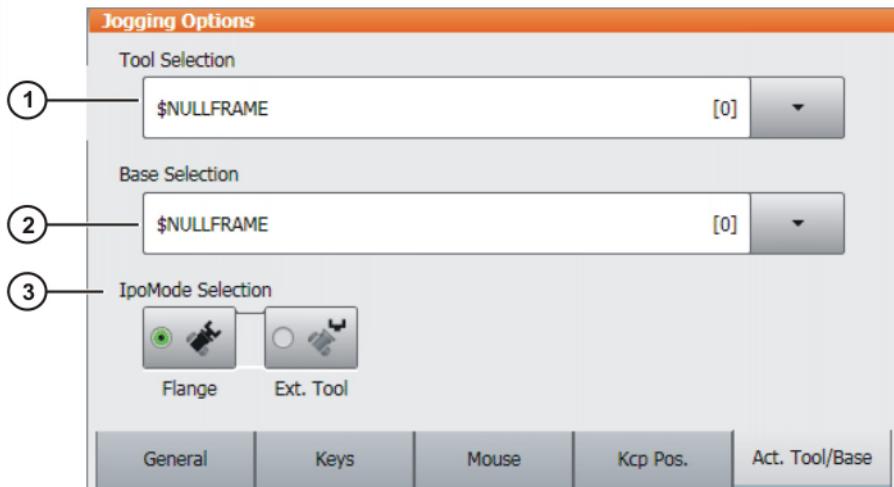


Fig. 4-18: Act. Tool/Base tab

Description	Item	Description
	1	The current tool is displayed here. A different tool can be selected. (>>> 4.15.4 "Selecting the tool and base" Page 71) The display <b>Unknown [?]</b> means that no tool has yet been calibrated.
	2	The current base is displayed here. A different base can be selected. (>>> 4.15.4 "Selecting the tool and base" Page 71) The display <b>Unknown [?]</b> means that no base has yet been calibrated.
	3	Select the interpolation mode: <ul style="list-style-type: none"> <li>■ <b>Flange:</b> The tool is mounted on the mounting flange.</li> <li>■ <b>ext. Tool:</b> The tool is a fixed tool.</li> </ul>

#### 4.15.2 Activating the jog mode

- Procedure**
1. Open the **Jogging Options** window.  
(>>> 4.15.1 “Jog options” window” Page 66)

2. To activate the jog mode “Jog keys”:  
On the **Keys** tab, activate the **Activate Keys** check box.  
To activate the jog mode “Space Mouse”:  
On the **Mouse** tab, activate the **Activate Mouse** check box.

- Description**
- Both jog modes “Jog keys” and “Space Mouse” can be activated simultaneously. If the robot is jogged using the keys, the Space Mouse is disabled until the robot comes to a standstill. If the Space Mouse is actuated, the keys are disabled.

#### 4.15.3 Setting the jog override (HOV)

- 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.

- Procedure**
1. Touch the **POV/HOV** status indicator. The **Overrides** window is opened.
  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%, 3%, 1%
    - Slider: The override can be adjusted in 1% steps.
  3. Touch the **POV/HOV** status indicator again. (Or touch the area outside the window.)
- The window closes and the selected override value is applied.



The **Jog options** window can be opened via **Options** in the **Overrides** window.

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

#### 4.15.4 Selecting the tool and base

- Description**
- One tool (TOOL coordinate system) and one base (BASE coordinate system) must be selected for Cartesian jogging.
- Procedure**
1. Touch the status indicator **Cur. tool/base**. The **Cur. tool/base** window opens.
  2. Select the desired tool and base.
  3. The window closes and the selection is applied.

#### 4.15.5 Axis-specific jogging with the jog keys

- Precondition**
- The jog mode “Jog keys” is active.
  - Operating mode T1
- Procedure**
1. Select **Axes** as the coordinate system for the jog keys.
  2. Set jog override.
  3. Hold down the enabling switch.  
Axes A1 to A6 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.



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

#### 4.15.6 Cartesian jogging with the jog keys

- Precondition**
- The jog mode “Jog keys” is active.
  - Operating mode T1
  - Tool and base have been selected.  
(>>> 4.15.4 "Selecting the tool and base" Page 71)
- Procedure**
1. Select **World, Base or Tool** as the coordinate system for the jog keys.
  2. Set jog override.
  3. 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
4. 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.

#### 4.15.7 Configuring the Space Mouse

##### Procedure

1. Open the **Jog options** window and select the **Mouse** tab.  
(>>> 4.15.1 "Jog options" window" Page 66)
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

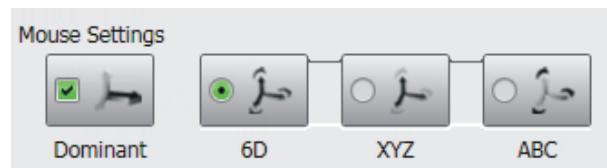


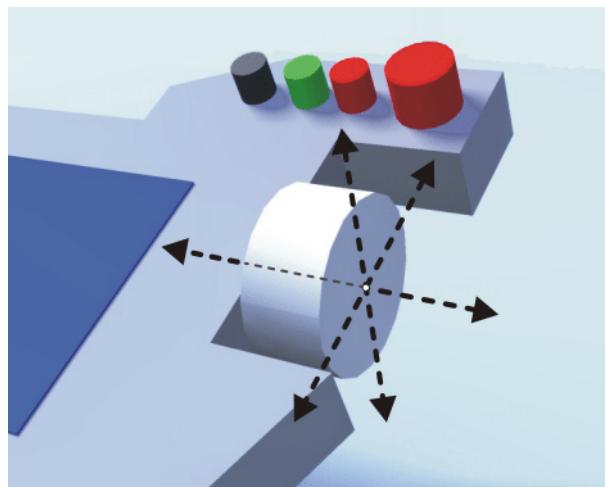
Fig. 4-19: Mouse settings

##### Dominant check box:

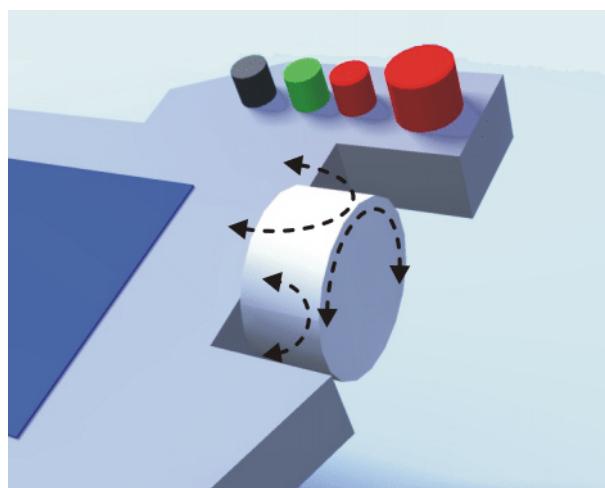
Depending on the dominant mode, the Space Mouse can be used to move just one axis or several axes simultaneously.

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
<b>6D</b>	<p>The robot can be moved by pulling, pushing, rotating or tilting the Space Mouse.</p> <p>The following motions are possible with Cartesian jogging:</p> <ul style="list-style-type: none"> <li>■ Translational motions in the X, Y and Z directions</li> <li>■ Rotational motions about the X, Y and Z axes</li> </ul>
<b>XYZ</b>	<p>The robot can only be moved by pulling or pushing the Space Mouse.</p> <p>The following motions are possible with Cartesian jogging:</p> <ul style="list-style-type: none"> <li>■ Translational motions in the X, Y and Z directions</li> </ul>
<b>ABC</b>	<p>The robot can only be moved by rotating or tilting the Space Mouse.</p> <p>The following motions are possible with Cartesian jogging:</p> <ul style="list-style-type: none"> <li>■ Rotational motions about the X, Y and Z axes</li> </ul>



**Fig. 4-20: Pushing and pulling the Space Mouse**



**Fig. 4-21: Rotating and tilting the Space Mouse**

#### 4.15.8 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°.

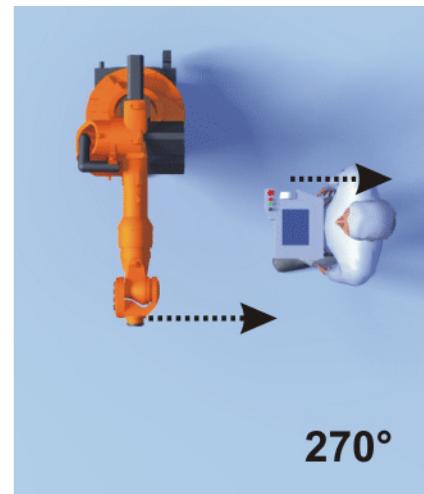
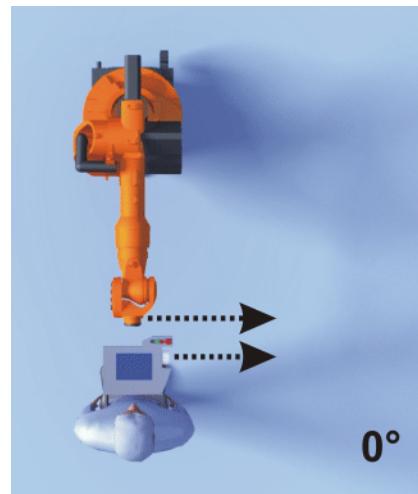


Fig. 4-22: 6D mouse: 0° and 270°

**Precondition**

- Operating mode T1

**Procedure**

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



Fig. 4-23: 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.

#### 4.15.9 Cartesian jogging with the Space Mouse

##### Precondition

- The jog mode "Space Mouse" is active.
- T1 mode
- Tool and base have been selected.  
(>>> 4.15.4 "Selecting the tool and base" Page 71)
- The Space Mouse is configured.  
(>>> 4.15.7 "Configuring the Space Mouse" Page 72)
- The alignment of the Space Mouse has been defined.  
(>>> 4.15.8 "Defining the alignment of the Space Mouse" Page 74)

##### Procedure

1. Select **World**, **Base** or **Tool** as the coordinate system for the Space Mouse.
2. Set the jog override.
3. Press and hold down the enabling switch.
4. 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.

#### 4.15.10 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. Incremental jogging is not possible in the case of jogging with the Space Mouse.

Areas of application:

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

The following options are available:

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

- The jog mode "Jog keys" is active.
- Operating mode T1

##### Procedure

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

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

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

## 4.16 Jogging external axes

<b>Description</b>	External axes cannot be moved using the Space Mouse. If “Space Mouse” mode is selected, only the robot can be jogged with the Space Mouse. The external axes, on the other hand, must be jogged using the jog keys.										
<b>Precondition</b>	<ul style="list-style-type: none"> <li>■ The jog mode “Jog keys” is active.</li> <li>■ Operating mode T1</li> </ul>										
<b>Procedure</b>	<ol style="list-style-type: none"> <li>1. Select the desired kinematics group, e.g. <b>External axes</b>, on the <b>Keys</b> tab in the <b>Jog options</b> window. The type and number of kinematics groups available depend on the system configuration.</li> <li>2. Set jog override.</li> <li>3. Hold down the enabling switch. The axes of the selected kinematics group are displayed next to the jog keys.</li> <li>4. Press the Plus or Minus jog key to move an axis in the positive or negative direction.</li> </ol>										
<b>Kinematic groups</b>	Depending on the system configuration, the following kinematics groups may be available.										
<table border="1"> <thead> <tr> <th><b>Kinematics group</b></th><th><b>Description</b></th></tr> </thead> <tbody> <tr> <td><b>Robot axes</b></td><td>The robot axes can be moved using the jog keys. The external axes cannot be jogged.</td></tr> <tr> <td><b>External axes</b></td><td>All configured external axes (e.g. external axes E1 to E5) can be moved using the jog keys.</td></tr> <tr> <td><b>NAME / External Kinematics Group n</b></td><td> <p>The axes of an external kinematics group can be moved using the jog keys.</p> <p>The name is taken from the system variable <code>\$ETn_NAME</code> (<math>n</math> = number of the external kinematics system). If <code>\$ETn_NAME</code> is empty, the default name <b>External Kinematics Group n</b> is displayed.</p> </td></tr> <tr> <td>[User-defined kinematics group]</td><td> <p>The axes of a user-defined kinematics group can be moved using the jog keys.</p> <p>The name corresponds to the name of the user-defined kinematics group.</p> </td></tr> </tbody> </table>		<b>Kinematics group</b>	<b>Description</b>	<b>Robot axes</b>	The robot axes can be moved using the jog keys. The external axes cannot be jogged.	<b>External axes</b>	All configured external axes (e.g. external axes E1 to E5) can be moved using the jog keys.	<b>NAME / External Kinematics Group n</b>	<p>The axes of an external kinematics group can be moved using the jog keys.</p> <p>The name is taken from the system variable <code>\$ETn_NAME</code> (<math>n</math> = number of the external kinematics system). If <code>\$ETn_NAME</code> is empty, the default name <b>External Kinematics Group n</b> is displayed.</p>	[User-defined kinematics group]	<p>The axes of a user-defined kinematics group can be moved using the jog keys.</p> <p>The name corresponds to the name of the user-defined kinematics group.</p>
<b>Kinematics group</b>	<b>Description</b>										
<b>Robot axes</b>	The robot axes can be moved using the jog keys. The external axes cannot be jogged.										
<b>External axes</b>	All configured external axes (e.g. external axes E1 to E5) can be moved using the jog keys.										
<b>NAME / External Kinematics Group n</b>	<p>The axes of an external kinematics group can be moved using the jog keys.</p> <p>The name is taken from the system variable <code>\$ETn_NAME</code> (<math>n</math> = number of the external kinematics system). If <code>\$ETn_NAME</code> is empty, the default name <b>External Kinematics Group n</b> is displayed.</p>										
[User-defined kinematics group]	<p>The axes of a user-defined kinematics group can be moved using the jog keys.</p> <p>The name corresponds to the name of the user-defined kinematics group.</p>										

## 4.17 Bypassing workspace monitoring

<b>Description</b>	Workspaces can be configured for a robot. These serve to protect the system. A workspace is always of one of the two following types:
	<ul style="list-style-type: none"> <li>■ Space that the robot must not leave A monitoring function is triggered if the robot leaves the space.</li> <li>■ Space that the robot must not enter A monitoring function is triggered if the robot enters the space.</li> </ul>

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

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

<b>Precondition</b>	<ul style="list-style-type: none"> <li>■ User group “Expert”</li> <li>■ Operating mode T1</li> </ul>
<b>Procedure</b>	<ol style="list-style-type: none"> <li>1. In the main menu, select <b>Configuration &gt; Miscellaneous &gt; Workspace monitoring &gt; Override</b>.</li> <li>2. Move the robot manually out of the violated space.</li> </ol> <p>Once the robot has left the violated space, the workspace monitoring is automatically active again.</p>

## 4.18 Display functions

### 4.18.1 Measuring and displaying energy consumption

<b>Description</b>	<p>The overall energy consumption of the robot and robot controller can be displayed on the smartHMI. A prerequisite for this is that measurement of energy consumption is possible for the robot type used.</p> <p>The energy consumption of optional robot controller components (e.g. US1, US2, etc.) and of other controllers is not taken into consideration. It is always the consumption for the last 60 minutes since the most recent cold start that is displayed. Furthermore, the user has the option of starting and stopping measurements manually.</p> <p>Traces can be made for the consumption values. The predefined configuration Tracedef_KRC_EnergyCalc is available for this.</p> <p>The data can also be transferred to a higher-level controller by means of PROFlenergy. PROFlenergy is a component of KR C4 PROFINET.</p> <p>There are two ways of starting and stopping measurements:</p> <ul style="list-style-type: none"> <li>■ In the <b>Energy consumption</b> window (<a href="#">&gt;&gt;&gt; Fig. 4-24</a> )</li> <li>■ Via KRL</li> </ul>
<b>Precondition</b>	<ul style="list-style-type: none"> <li>■ Measurement of energy consumption is possible for the robot type used. If not, the boxes in the <b>Energy consumption</b> window are grayed out.</li> </ul>
<b>Procedure</b>	<p>Starting and stopping a measurement in the <b>Energy consumption</b> window:</p> <ol style="list-style-type: none"> <li>1. In the main menu, select <b>Display &gt; Energy consumption</b>. The <b>Energy consumption</b> window opens.</li> <li>2. If required, activate the check box next to <b>Refresh</b>.</li> <li>3. Press <b>Start measuring</b>. A red dot to the right of the top line now indicates that a measurement is in progress.</li> <li>4. To stop the measurement, press <b>Stop measuring</b>. The result is displayed.</li> </ol> <p>Starting and stopping a measurement via KRL:</p> <ol style="list-style-type: none"> <li>1. Start the measurement via <code>\$ENERGY_MEASURING.ACTIVE = TRUE</code> (possible via the KRL program or variable correction function). The measurement starts.</li> <li>2. In the main menu, select <b>Display &gt; Energy consumption</b>. The <b>Energy consumption</b> window opens. A red dot to the right of the top line indicates the measurement that is in progress.</li> <li>3. If required, activate the check box next to <b>Refresh</b>.</li> </ol>

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

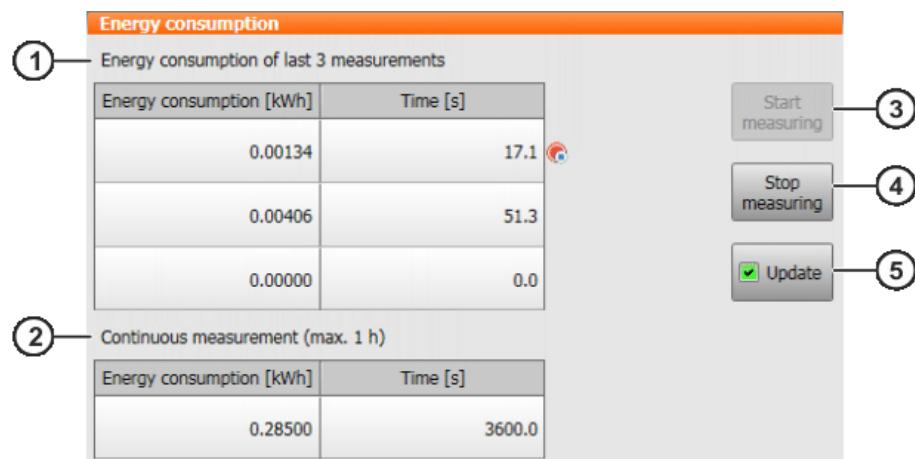


Fig. 4-24: “Energy consumption” window

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. <b>Start measuring</b> is not available if a measurement is currently active.

Item	Description
4	Stops an active measurement. How the measurement was started (by means of <b>Start measuring</b> or via KRL) is irrelevant.
5	<ul style="list-style-type: none"> <li>■ Check box active: While a measurement is being carried out, the result display is continually refreshed.</li> <li>■ 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.</li> </ul>

#### 4.18.2 Displaying the actual position

- Procedure**
1. In the main menu, select **Display > Actual position**. The Cartesian actual position is displayed.
  2. To display the axis-specific actual position, press **Axis-specific**.
  3. To display the Cartesian actual position again, press **Cartesian**.

**Description**

**Actual position, Cartesian:**

The current position (X, Y, Z) and orientation (A, B, C) of the TCP are displayed. Status and Turn are also displayed.

**Actual position, axis-specific:**

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

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

Robot Position (Axis Specific)			
Axis	Pos. [deg, mm]	Motor [deg]	
A1	0.00	0.00	
A2	-90.00	22530.00	
A3	90.00	-24228.95	
A4	0.00	0.00	
A5	0.00	0.00	
A6	0.00	0.00	
E1	0.00	0.00	

Cartesian

Fig. 4-25: Actual position, axis-specific

#### 4.18.3 Displaying digital inputs/outputs

- Procedure**
1. In the main menu, select **Display > Inputs/outputs > Digital I/O**.
  2. To display a specific input/output:
    - 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 input/output with this number.

**Description**

No.	Value	State	Name
6	○		Eingang
7	●	SIM	Eingang
8	○		Eingang
9	○		Eingang
10	○	SYS	Eingang
11	○		Finnann

**Fig. 4-26: Digital inputs**

No.	Value	State	Name
138	○		Ausgang
139	○		Ausgang
140	○		Ausgang
141	●		Ausgang
142	○		Ausgang
143	○		Ausgang

**Fig. 4-27: 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	<b>SIM:</b> The input/output is simulated. <b>SYS:</b> The value of the input/output is saved in a system variable. This input/output is write-protected.
4	Name of the input/output

The following buttons are available:

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.

Button	Description
<b>Value</b>	Toggles the selected input/output between TRUE and FALSE. Precondition: The enabling switch is pressed. <ul style="list-style-type: none"> <li>■ <b>Value</b> is not available in EXT mode.</li> <li>■ <b>Value</b> is only available for inputs if simulation is activated.</li> </ul>
<b>Name</b>	The name of the selected input or output can be changed.

#### 4.18.4 Displaying analog inputs/outputs

##### Procedure

1. In the main menu, select **Display > Inputs/outputs > Analog I/O**.
2. To display a specific input/output:
  - 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 input/output with this number.

The following buttons are available:

Button	Description
<b>Go to</b>	The number of the input or output being searched for can be entered.
<b>Voltage</b>	A voltage can be entered for the selected output. <ul style="list-style-type: none"> <li>■ <b>-10 ... 10 V</b></li> </ul> This button is only available for outputs.
<b>Name</b>	The name of the selected input or output can be changed.

#### 4.18.5 Displaying inputs/outputs for Automatic External

##### Procedure

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

##### Description

Automatic External - Monitor: Inputs					
St.	Term	Type	Name	Value	
1	0 current programno.	Var	PGNO	0	
2	Type programno.	AI	PGNO_TYPE	1	
3	Bitwidth programno.	AI	PGNO_LENGTH	8	
4	First bit programno.	AI	PGNO_FBIT	33	
5	Parity bit	AI	PGNO_PARITY	41	
6	Programno. valid	AI	PGNO_VALID	42	
7	Programstart	AI	\$EXT_START	1026	
8	Move enable	AI	\$MOVE_ENABLE	1025	
9	Error confirmation	AI	\$CONF_MESS	1026	
10	Drives off (invers)	AI	\$DRIVES_OFF	1025	
11	Drives on	AI	\$DRIVES_ON	140	
12	Activate interface	AI	\$I_O_ACT	1025	

Fig. 4-28: Automatic External inputs (detail view)

Automatic External - Monitor: Outputs					
	St.	Term	Type	Name	Value
1	●	Control ready	MO	\$RC_RDY1	137
2	●	Alarm stop active	MO	\$ALARM_STOP	1013
3	●	User safety switch closed	MO	\$USER_SAF	1011
4	●	Drives ready	MO	\$PERI_RDY	1012
5	●	Robot calibrated	MO	\$ROB_CAL	1001
6	○	Interface active	MO	\$I_O_ACTCONF	140
7	○	Error collection	MO	\$STOPMESS	1010
8	●	Internal emergency stop	MO	IntEstop	1002

Fig. 4-29: Automatic External outputs (detail view)

Item	Description
1	Number
2	Status <ul style="list-style-type: none"> <li>■ Gray: inactive (FALSE)</li> <li>■ Red: active (TRUE)</li> </ul>
3	Long text name of the input/output
4	Type <ul style="list-style-type: none"> <li>■ Green: input/output</li> <li>■ Yellow: variable or system variable (\$...)</li> </ul>
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.

The following buttons are available:

Button	Description
<b>Config.</b>	Switches to the configuration of the Automatic External interface. ( <b>&gt;&gt;&gt; 6.19.2 "Configuring Automatic External inputs/outputs" Page 197</b> )
<b>Inputs/Outputs</b>	Toggles between the windows for inputs and outputs.
<b>Details/Normal</b>	Toggles between the <b>Details</b> and <b>Normal</b> views.

#### 4.18.6 Displaying and modifying the value of a variable

This function is also called “variable correction”.

##### Precondition

To modify a variable:

- “Expert” user group

##### 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

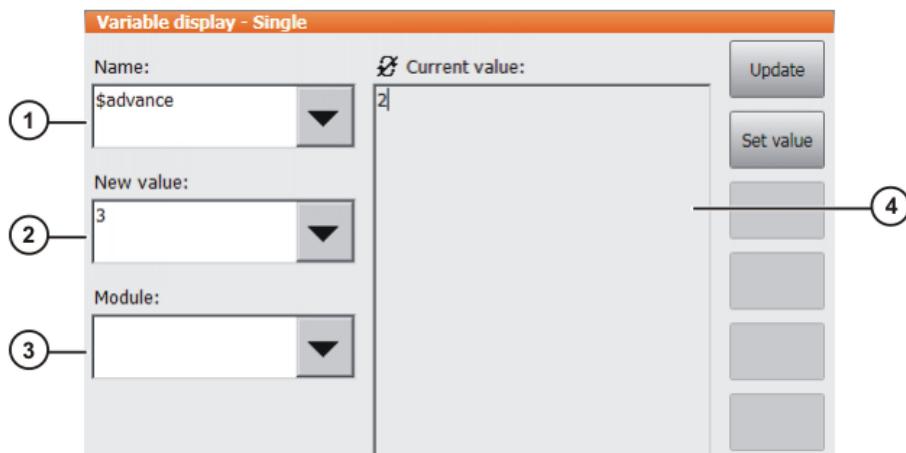


Fig. 4-30: Variable Overview - Single window

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 <b>Module</b> box is irrelevant.
4	This box has two states: <ul style="list-style-type: none"> <li>■  : The displayed value is not refreshed automatically.</li> <li>■  : The displayed value is refreshed automatically.</li> </ul> Switching between the states: <ul style="list-style-type: none"> <li>■ Press <b>Refresh</b>.</li> </ul>

#### 4.18.7 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.

#### 4.18.8 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 174)

<b>Precondition</b>	The minimum user group required for displaying and/or modifying variables depends on the configuration settings for the variable overview.
<b>Procedure</b>	<ol style="list-style-type: none"> <li>1. In the main menu, select <b>Display &gt; Variable &gt; Overview &gt; Display</b>. The <b>Variable overview – Display</b> window is opened.</li> <li>2. Select the desired group.</li> <li>3. Select the cell to be modified. Carry out modification using the buttons.</li> <li>4. Press <b>OK</b> to save the change and close the window.</li> </ol>

#### Description

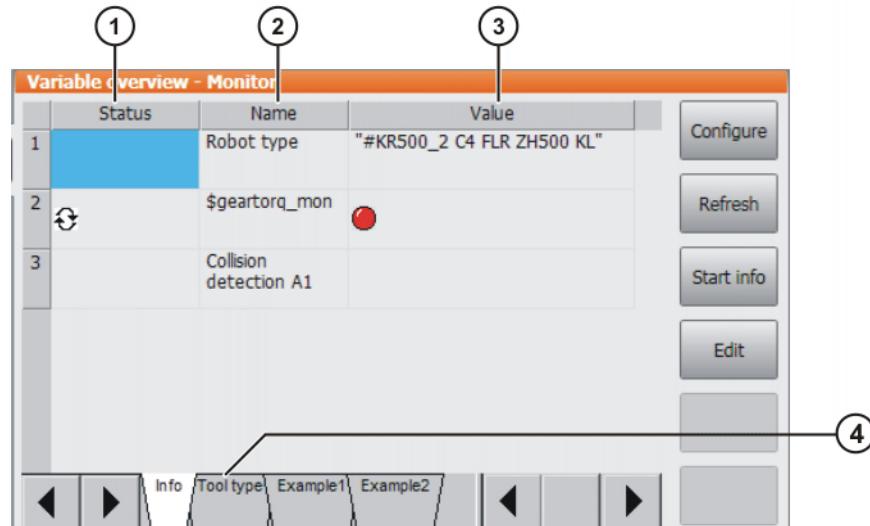


Fig. 4-31: Variable overview - Monitor window

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

Item	Description
3	Value of the variable. In the case of inputs/outputs, the state is indicated: <ul style="list-style-type: none"> <li>■ <b>Gray:</b> inactive (FALSE)</li> <li>■ <b>Red:</b> active (TRUE)</li> </ul>
4	There is one tab per group.

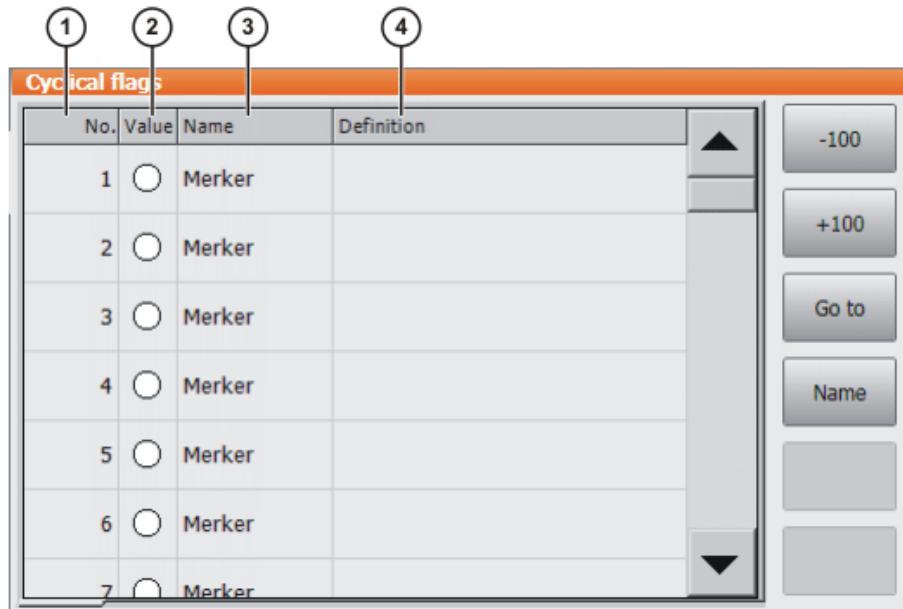
The following buttons are available:

Button	Description
<b>Config.</b>	Switches to the configuration of the variable overview.  (>>> 6.10 "Configuring the variable overview" Page 174)  This button is not available in the user group "User".
<b>Refresh all</b>	Refreshes the display.
<b>Cancel info</b>	Deactivates the automatic refreshing function.
<b>Start info</b>	Activates the automatic refreshing function.  A maximum of 12 variables per group can be refreshed automatically.
<b>Edit</b>	Switches the current cell to edit mode so that the name or value can be modified. In the <b>Value</b> column, this button changes the state of inputs/outputs (TRUE/FALSE).  This button is only available in the user group "User" if it has been enabled in the configuration.  <b>Note:</b> The values of write-protected variables cannot be changed.

#### 4.18.9 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. 4-32: Cyclical flags**

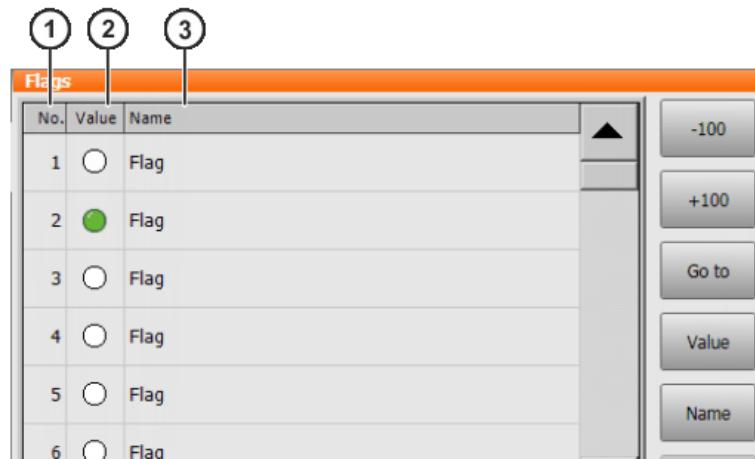
<b>Item</b>	<b>Description</b>
1	Flag number
2	Value of the flag. The icon is green if a flag is set.
3	Name of the flag
4	The conditions linked to the setting of a cyclical flag are indicated here.

The following buttons are available:

<b>Button</b>	<b>Description</b>
<b>-100</b>	Toggles back 100 flags in the display.
<b>+100</b>	Toggles forward 100 flags in the display.
<b>Go to</b>	The number of the flag being searched for can be entered.
<b>Name</b>	The name of the selected flag can be modified.

**4.18.10 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. 4-33: 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

The following buttons are available:

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.

**4.18.11 Displaying counters****Procedure**

1. In the main menu, select **Display > Variable > Counter**. The **Counter** window is opened.
2. To display a specific counter:
  - 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 counter with this number.

**Description**

No.	Value	Name
1	0	Zaehler
2	0	Zaehler
3	0	Zaehler
4	8	my_counter
5	0	Zaehler
6	0	Zaehler
7	0	Zaehler

**Fig. 4-34: Counter**

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

The following buttons are available:

Button	Description
<b>Go to</b>	The number of the counter being searched for can be entered.
<b>Value</b>	A value can be entered for the selected counter.
<b>Name</b>	The name of the selected counter can be modified.

**4.18.12 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**

No.	Status	T	Value [m...]	Name
1	■		0	Timer
2	■		0	Timer
3	■	✓	59376	Timer
4	■		0	Timer
5	■		0	Timer
6	■		0	Timer
7	■		0	Timer

**Fig. 4-35: Timer**

Item	Description
1	Number of the timer
2	Status of the timer <ul style="list-style-type: none"> <li>■ If the timer is activated, this is indicated in green.</li> <li>■ If the timer is deactivated, this is indicated in red.</li> </ul>
3	State of the timer <ul style="list-style-type: none"> <li>■ If the value of the timer is &gt; 0, the timer flag is set (red check mark).</li> <li>■ If the value of the timer is ≤ 0, no timer flag is set.</li> </ul>
4	Value of the timer (unit: ms)
5	Name of timer

The following buttons are available:

Button	Description
<b>Go to</b>	The number of the timer being searched for can be entered.
<b>State</b>	Toggles the selected timer between TRUE and FALSE. Precondition: The enabling switch is pressed.
<b>Value</b>	A value can be entered for the selected timer.
<b>Name</b>	The name of the selected timer can be modified.

**4.18.13 Displaying calibration data****Procedure**

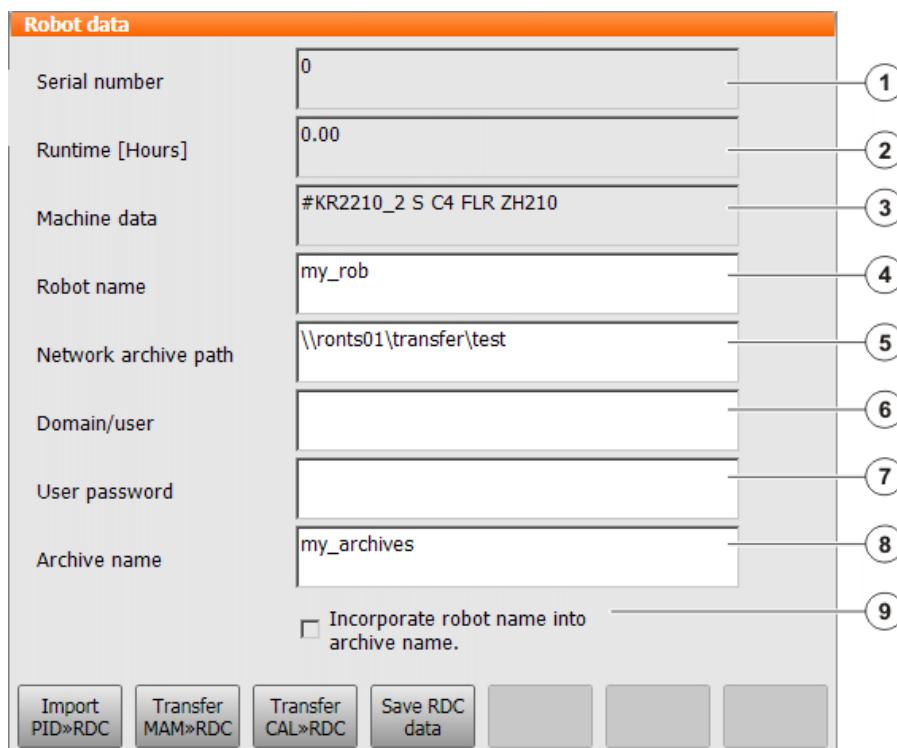
- In the main menu, select **Start-up > Calibrate > Calibration points** and the desired menu item:
  - **Tool type**
  - **Base type**
  - **External axis**
- Enter the number of the tool, base or external kinematic system.  
The calibration method and the calibration data are displayed.

#### 4.18.14 Displaying information about the robot and robot controller

<b>Procedure</b>	<ul style="list-style-type: none"><li>■ In the main menu, select <b>Help &gt; Info</b>.</li></ul>														
<b>Description</b>	The information is required, for example, when requesting help from KUKA Customer Support.  The tabs contain the following information:														
<table border="1"><thead><tr><th>Tab</th><th>Description</th></tr></thead><tbody><tr><td><b>Info</b></td><td><ul style="list-style-type: none"><li>■ Robot controller type</li><li>■ Robot controller version</li><li>■ User interface version</li><li>■ Kernel system version</li></ul></td></tr><tr><td><b>Robot</b></td><td><ul style="list-style-type: none"><li>■ Robot name</li><li>■ Robot type and configuration</li><li>■ Service life</li></ul><p>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 \$ROB-RUNTIME.</p><ul style="list-style-type: none"><li>■ Number of axes</li><li>■ List of external axes</li><li>■ Machine data version</li></ul></td></tr><tr><td><b>System</b></td><td><ul style="list-style-type: none"><li>■ Control PC name</li><li>■ Operating system version</li><li>■ Storage capacities</li></ul></td></tr><tr><td><b>Options</b></td><td>Additionally installed options and technology packages</td></tr><tr><td><b>Comments</b></td><td>Additional comments</td></tr><tr><td><b>Modules</b></td><td>Names and versions of important system files  The <b>Export</b> button exports the contents of the <b>Modules</b> tab to the file C:\KRC\ROBOTER\LOG\FILEVERSIONS.TXT.</td></tr></tbody></table>		Tab	Description	<b>Info</b>	<ul style="list-style-type: none"><li>■ Robot controller type</li><li>■ Robot controller version</li><li>■ User interface version</li><li>■ Kernel system version</li></ul>	<b>Robot</b>	<ul style="list-style-type: none"><li>■ Robot name</li><li>■ Robot type and configuration</li><li>■ Service life</li></ul> <p>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 \$ROB-RUNTIME.</p> <ul style="list-style-type: none"><li>■ Number of axes</li><li>■ List of external axes</li><li>■ Machine data version</li></ul>	<b>System</b>	<ul style="list-style-type: none"><li>■ Control PC name</li><li>■ Operating system version</li><li>■ Storage capacities</li></ul>	<b>Options</b>	Additionally installed options and technology packages	<b>Comments</b>	Additional comments	<b>Modules</b>	Names and versions of important system files  The <b>Export</b> button exports the contents of the <b>Modules</b> tab to the file C:\KRC\ROBOTER\LOG\FILEVERSIONS.TXT.
Tab	Description														
<b>Info</b>	<ul style="list-style-type: none"><li>■ Robot controller type</li><li>■ Robot controller version</li><li>■ User interface version</li><li>■ Kernel system version</li></ul>														
<b>Robot</b>	<ul style="list-style-type: none"><li>■ Robot name</li><li>■ Robot type and configuration</li><li>■ Service life</li></ul> <p>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 \$ROB-RUNTIME.</p> <ul style="list-style-type: none"><li>■ Number of axes</li><li>■ List of external axes</li><li>■ Machine data version</li></ul>														
<b>System</b>	<ul style="list-style-type: none"><li>■ Control PC name</li><li>■ Operating system version</li><li>■ Storage capacities</li></ul>														
<b>Options</b>	Additionally installed options and technology packages														
<b>Comments</b>	Additional comments														
<b>Modules</b>	Names and versions of important system files  The <b>Export</b> button exports the contents of the <b>Modules</b> tab to the file C:\KRC\ROBOTER\LOG\FILEVERSIONS.TXT.														

#### 4.18.15 Displaying/editing robot data

<b>Precondition</b>	<ul style="list-style-type: none"><li>■ T1 or T2 operating mode</li><li>■ No program is selected.</li></ul>
<b>Procedure</b>	<ul style="list-style-type: none"><li>■ In the main menu, select <b>Start-up &gt; Robot data</b>.</li></ul>

**Description****Fig. 4-36: 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	Robot name. The robot 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 <b>Incorporate robot name into archive name.</b> is not activated. A name for the archive file can be defined here.
9	<ul style="list-style-type: none"> <li>■ <b>Check box active:</b> The robot name is used as the name for the archive file. If no robot name is defined, the name <i>archive</i> is used.</li> <li>■ <b>Check box not active:</b> A separate name can be defined for the archive file.</li> </ul>

The following buttons are available in the user group "Expert":

Button	Description
<b>Transfer PID&gt;&gt;RDC</b>	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.
<b>Transfer MAM&gt;&gt;RDC</b>	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.
<b>Transfer CAL&gt;&gt;RDC</b>	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.
<b>Save RDC data</b>	The data on the RDC can be backed up temporarily in the directory C:\KRC\Roboter\RDC by pressing this button.  <b>Note:</b> The directory is deleted when the robot controller is rebooted or data are archived. If the RDC data are to be retained permanently, they must be backed up elsewhere.

#### 4.19 Resetting safe input/output errors

**Description** Error messages relating to safe inputs/outputs are not automatically deactivated when the cause of the error has been eliminated. They must be reset by means of an explicit operator action. This can be done using the menu command **Reset safety I/O error**. If a program has been selected, it can remain selected.

If the command is executed while no corresponding message is active, it has no effect.

The following messages can be reset in this way:

Number	Message
15000	<i>Main contactor: loop back contact error</i>
15004	<i>I/O error in safe device</i>
15006	<i>Peripheral contactor: loop back contact error</i>
15012	<i>Enabling switch error</i>
15095	<i>SIB/SIB-Extended: loop back contact error</i>

- Precondition**
- T1 or T2 mode
  - The cause of the error has been eliminated.

- Procedure**
1. In the main menu, select **Start-up > Service > Reset safety I/O error**.  
The robot controller resets the message associated with the error and generates an acknowledgement message.
  2. Acknowledge the message. This is only possible in modes T1 and T2.  
It is now possible to select a different operating mode and restart the system.

## 4.20 Displaying the battery state

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

The battery charge is displayed by means of the system variable \$ACCU\_STATE.

The state can only be displayed and not modified.

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

**States** The following tables indicate the possible states of \$ACCU\_STATE.

The user must configure the info to the PLC himself.



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

### #CHARGE\_OK

**Meaning:** The charging current dropped as required after booting and/or the battery tested positive in the battery test.

**Action required by the user:** Do not exchange the battery.

**Info to PLC:** Supply voltage disconnection OK.

**Message:** No message.

### #CHARGE\_OK\_LOW

**Meaning:** The charging current dropped as required after booting and/or the battery tested positive in the battery test. The battery is not fully charged, however, after the maximum charging time.

**Action required by the user:** Exchange the battery.

**Info to PLC:** Supply voltage disconnection OK.

**Message:** Battery warning - full charge not possible

### #CHARGE\_UNKNOWN

**Meaning:** The battery is being charged. Or the battery has not yet been checked since the controller was booted. Or the charging current has not yet dropped sufficiently.

**Action required by the user:** Do not exchange the battery.

**Info to PLC:** Supply voltage disconnection can cause errors in Hibernate mode.

**Message:** No message

### #CHARGE\_TEST\_NOK

**Meaning:** The result of the battery test was negative.

**Action required by the user:** Exchange the battery.

**Info to PLC:** Supply voltage disconnection can cause errors in Hibernate mode.

**Message:** Battery defective - load test failed

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

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

## 5 Start-up and recommissioning

### 5.1 Protecting the Windows system

KUKA recommends the following measures for protecting the Windows system:

Measure	Description
Use <b>KUKA.NonAdmin</b>	Add-on functionality for protection of the Windows system against unwanted access
Use <b>KUKA.CPC</b> or <b>KUKA.Ikarus</b>	KUKA.CPC: Add-on functionality for protection of the Windows system against unauthorized access, protection against malicious software
	KUKA.Ikarus: Add-on functionality for protection against malicious software
Change the default password	(>>> 5.2 "Changing the password for a Windows system" Page 95)



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

### 5.2 Changing the password for a Windows system

#### Description

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

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

The password can be changed using the procedure described here.

Alternatively, the password can be changed via WorkVisual. If KUKA.NonAdmin is installed on the robot controller, the password can be changed exclusively via WorkVisual.

There are no restrictions for the password, e.g. in terms of length or complexity.

#### NOTICE

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

#### Precondition

- KUKA.NonAdmin is not installed on the robot controller.
- Administrator rights

#### Procedure

1. Open the Windows Start menu and select **Run....**

2. Enter the `cmd` command in the **Open** box and confirm with the Enter key.  
The command window opens.

3. Enter the following command:

```
c:\krc\util\krcuserpw\changepwd.exe /u=kukauser  
/op=OLD_PW /p=NEW_PW /cp
```

Here, enter the current password instead of `OLD_PW` and the desired new password instead of `NEW_PW`.

4. Confirm by pressing the Enter key.

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

**Parameters**

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

**Log file**

The change is logged in:

- C:\KRC\ROBOTER\LOG\\_ChangePwd.log

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

### 5.2.1 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_KUKACONFIG	0x0020 The VxWorks user cannot be read from kuka.config* for “Change password”.
ERR_READINGPASSWORD_KUKACONFIG	0x0040 The VxWorks password cannot be read from kuka.config* for “Change password”.
ERR_EXTRACTINGINFO_STARTKRC	0x0080 The following data cannot be read from Start-Krc.KEC in the path C:\KRC for “Change password”: <ul style="list-style-type: none"> <li>■ User name, domain and password of the KEC user</li> </ul>
ERR_WITETARGETPASSWORD	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 in the path C:\Windows\System32 for “Change password”.
ERR_COMMANDINPUTFILEACCESSError	0x1000 There is no response file with the options for the password change or it cannot be accessed.

<b>Code</b>		<b>Description</b>
ERR_WRONG_NONADMINPACK AGE	0x2000	The installed NonAdmin version is not suitable for the password change.
ERR_WRONG_CPCPACKAGE	0x4000	The installed CPC version is not suitable for the password change.

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

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

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



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

### 5.3 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.

### 5.4 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.



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

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**



Information about the warm-up parameters can be found here:  
(>>> 6.15.2 "Configuring warm-up" Page 185)

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

**Precondition**

- User group "Expert"
- Operating mode T1 or T2.
- No program is selected.

**Procedure**

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

## 5.5 Defining hardware options

**Precondition**

- User group "Safety maintenance"
- Operating mode T1 or T2

**Procedure**

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



**WARNING** Following modifications to the safety configuration, the values for the safe axis monitoring functions must be checked.  
(>>> 6.5 "Checking the values for the safe axis monitoring functions"  
Page 170)

**Description**

Parameter	Description
Customer interface	Select here which interface is used: <ul style="list-style-type: none"><li>■ <b>Automatic</b></li><li>■ <b>SIB with operating mode output</b></li></ul>

Parameter	Description
Input signal for peripheral contactor (US2)	<ul style="list-style-type: none"> <li>■ <b>Deactivated:</b> The peripheral contactor is not used. (Default)</li> <li>■ <b>By external PLC:</b> The peripheral contactor is switched by an external PLC via input US2.</li> <li>■ <b>By KRC:</b> The peripheral contactor is switched in accordance with the motion enable. If motion enable is present, the contactor is energized.</li> </ul> <p><b>Notes:</b></p> <ul style="list-style-type: none"> <li>■ For robot controllers with peripheral contactors and the “UL” option, this parameter must be set to <b>By KRC</b>.</li> <li>■ For robot controllers with no peripheral contactors, this parameter is deactivated (default setting) and is not displayed.</li> </ul> <p>The system variable \$US2_VOLTAGE_ON indicates the status of the peripheral voltage US2:</p> <ul style="list-style-type: none"> <li>■ TRUE: Voltage is switched on.</li> <li>■ FALSE: Voltage is switched off.</li> </ul>
Operator safety acknowledgement	<p>If the “Operator Safety” signal is lost and set again in Automatic mode, it must be acknowledged before operation can be continued.</p> <ul style="list-style-type: none"> <li>■ <b>By acknowledgement button:</b> 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.</li> <li>■ <b>External unit:</b> Acknowledgement is given by the system PLC.</li> </ul>

## 5.6 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”
- Operating mode T1 or T2



If one of the options KUKA.SafeOperation or KUKA.SafeRangeMonitoring is installed on the robot controller, different user groups may apply. Information can be found in the documentation for these options.

**Procedure**

1. In the main menu, select **Configuration > Safety configuration**.
2. Press **Device management**.
3. In the column **New safety ID**, press the ID to be modified and change the ID.
4. Press **Apply safety IDs**.
5. A request for confirmation is displayed, asking if the change should be saved. Confirm the request with **Yes**.
6. A message is displayed, indicating that the change has been saved. Confirm the message with **OK**.



This procedure can only be used to save changes to the safety ID. If other unsaved changes have been made elsewhere in the safety configuration, these are not saved here.

If an attempt is now made to close the safety configuration, a query is generated asking whether you wish to reject the changes or cancel the action. To save the changes, proceed as follows:

1. Cancel the action.
2. In the safety configuration, press **Save**. (If the **Save** button is not available, first go back a level by pressing **Back**.)
3. A request for confirmation is displayed, asking if all the changes should be saved. Confirm the request with **Yes**.
4. A message is displayed, indicating that the change has been saved. Confirm the message with **OK**.

All changes in the safety configuration are saved.



**WARNING** Following modifications to the safety configuration, the values for the safe axis monitoring functions must be checked.

(>>> 6.5 "Checking the values for the safe axis monitoring functions" Page 170)

## 5.7 Jogging the robot without a higher-level safety controller

### Description

To jog the robot without a higher-level safety controller, Start-up mode must first be activated. The robot can then be jogged in T1 mode.



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

(>>> 3.8.3.2 "Start-up mode" Page 37)

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:

System Software 8.2 or earlier: The robot controller automatically deactivates Start-up mode if it is no longer the case that all input signals at the discrete interface (and, if used, at the discrete safety interface for safety options) have the state "logic zero".

From System Software 8.3 onwards, on the other hand, 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 VKR C4: E2/E22 is closed.

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



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

#### Precondition

- T1 mode
- In the case of VKR C4: no E2/E22/E7 signals are activated via a USB stick or retrofit interface.
- In the case of RoboTeam: the local smartPAD is used.
- If the Ethernet safety interface is used: No connection to a higher-level safety controller
- If a discrete safety interface is used:  
For System Software 8.2 only: all input signals have the state “logic zero”.  
If the additional discrete safety interface for safety options is used, the inputs there must also have the state “logic zero”.  
(From System Software 8.3 onwards, Start-up mode is not dependent on the state of these inputs.)

#### Procedure

- In the main menu, select **Start-up > Service > Start-up mode**.

Menu	Description
<input checked="" type="checkbox"/> Start-up mode	Start-up mode is active. Touching the menu item deactivates the mode.
<input type="checkbox"/> Start-up mode	Start-up mode is not active. Touching the menu item activates the mode.

## 5.8 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**).

**5.9 Activating palletizing mode****Description**

Only relevant for palletizing robots with 6 axes!

In the case of palletizing robots with 6 axes, palletizing mode is deactivated by default and must be activated. When palletizing mode is active, A4 is locked at 0° and the mounting flange is parallel to the floor.

**Precondition**

- The robot is mastered.
- There is no load on the robot; i.e. there is no tool, workpiece or supplementary load mounted.

**Procedure**

- Activate palletizing mode in the program as follows:

```
$PAL_MODE = TRUE
```

**Alternative procedure**

1. Set \$PAL\_MODE to TRUE via the variable correction function.
2. The following message is displayed: *Palletizing mode: Move axis A4 [direction] into position.*  
Move A4 in the direction specified in the message (plus or minus).
3. Once A4 has reached its position (0°), the following message is displayed: *Palletizing mode: Move axis A5 [direction] into position.*  
Move A5 in the direction specified in the message (plus or minus).  
Once A5 has reached its position (90°), the message disappears.

The labels **A4** and **A5** next to the jog keys now disappear. These axes can no longer be jogged.

**Restrictions**

- After every cold restart of the robot controller, \$PAL\_MODE is automatically set to FALSE.



Recommendation: Integrate \$PAL\_MODE = TRUE into the initialization section of all programs for the palletizing robot.

- In the case of robots with palletizing mode active, payload determination with KUKA.LoadDataDetermination is not possible.



In the case of robots with palletizing mode active, payload determination with KUKA.LoadDataDetermination must not be carried out. Injuries or damage to property may result.

- If palletizing mode is active, the robot cannot be mastered. If mastering is nonetheless required, proceed as follows:
  - a. Remove all loads from the robot.
  - b. Set \$PAL\_MODE to FALSE via the variable correction function.
  - c. Master the robot.
  - d. Set \$PAL\_MODE to TRUE.  
(Not necessary if \$PAL\_MODE = TRUE is in the initialization section of all programs for the palletizing robot.)
  - e. Move the robot to the palletizing position.
  - f. Re-attach all loads to the robot.

## 5.10 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. 5-1: 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	(>> 5.10.8 "Reference mastering" Page 115)

Case	Comment
When the robot has been moved without the robot controller (e.g. with the release device)	- - -
After exchanging a gear unit After an impact with an end stop at more than 250 mm/s	Before carrying out a new mastering procedure, the old mastering data must first be deleted! Mastering data are deleted by manually unmastering the axes.  (>>> 5.10.10 "Manually unmastering axes" Page 123)
After a collision.	

### 5.10.1 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 <b>SEMD</b> (Standard Electronic Mastering Device)  Protective cap with fine thread, M20	Mastering with the probe, type <b>SEMD</b> (>>> 5.10.5 "Mastering with the SEMD" Page 109)  Mastering with the dial gauge (>>> 5.10.6 "Mastering with the dial gauge" Page 114)  Reference mastering  Only for mastering after certain maintenance work  (>>> 5.10.8 "Reference mastering" Page 115)
Gauge cartridge for <b>MEMD</b> (Micro Electronic Mastering Device)  Protective cap with fine thread, M8	Mastering with the probe, type <b>MEMD</b> On A6 in certain cases: mastering to the mark  (>>> 5.10.9 "Mastering with the MEMD and mark" Page 116)

#### SEMD/MEMD

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



**Fig. 5-2: Mastering kit with SEMD and MEMD**

- |                        |          |
|------------------------|----------|
| 1 Mastering box        | 4 SEMD   |
| 2 Screwdriver for MEMD | 5 Cables |
| 3 MEMD                 |          |

The thinner cable is the signal cable. It connects the SEMD or MEMD to the mastering box.

The thicker cable is the EtherCAT cable. It is connected to the mastering box and to the robot at X32.

**NOTICE**

- Leave the signal cable connected to the mastering box and disconnect it as little as possible. The pluggability of the M8 sensor connector is limited. Frequent connection/disconnection can result in damage to the connector.
- In the case of probes to which the signal cable is not permanently attached, always screw the device onto the gauge cartridge without the signal cable. Only then may the cable be attached to the device. Otherwise, the cable could be damaged. Similarly, when removing the device, the signal cable must always be removed from the device first. Only then may the device be removed from the gauge cartridge.
- After mastering, remove the EtherCAT cable from connection X32. Failure to do so may result in interference signals or damage.

### 5.10.2 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.

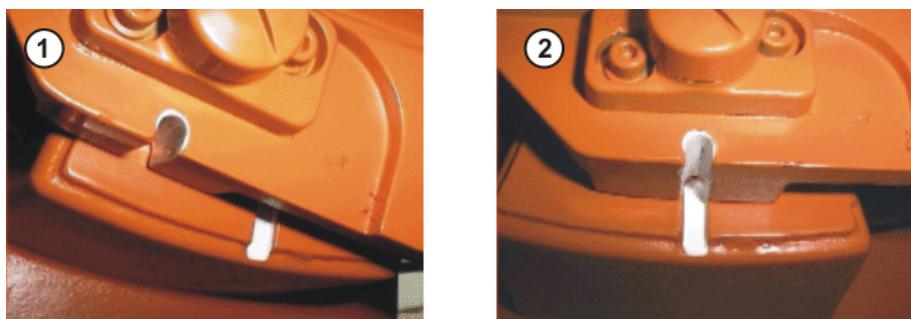


Fig. 5-3: Moving an axis to the pre-mastering position

**i** 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.  
(>>> 5.10.3 "Moving axes to the pre-mastering position using the probe" Page 107)

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.

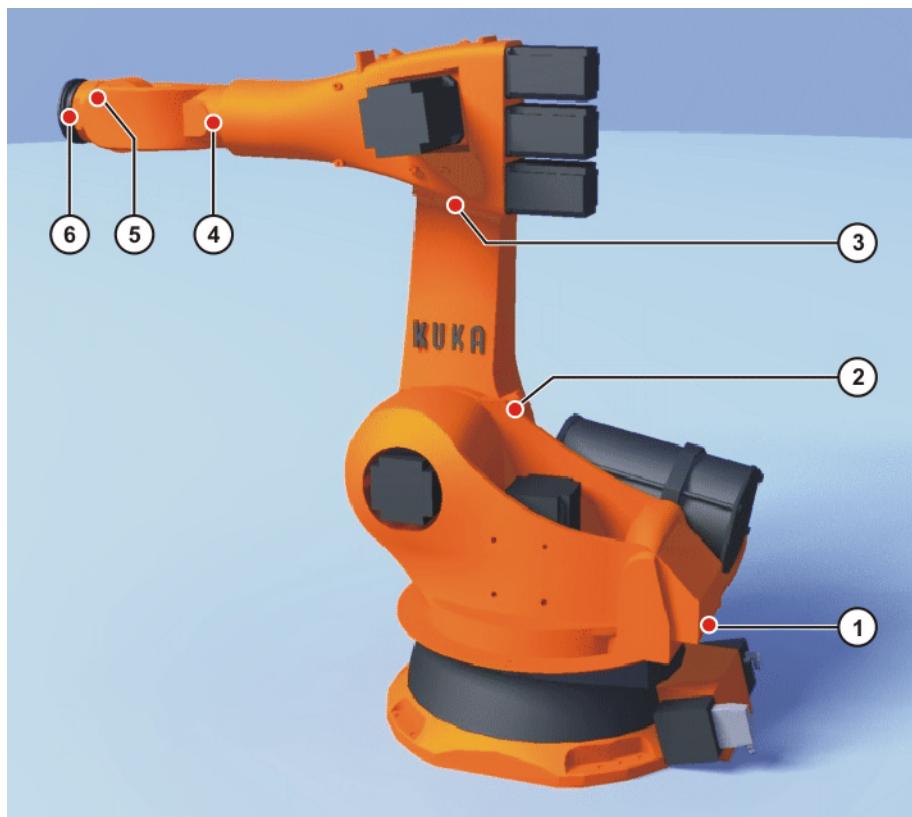


Fig. 5-4: Mastering marks on the robot

#### Precondition

- The jog mode "Jog keys" is active.
- Operating mode T1

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

#### Procedure

1. Select **Axes** as the coordinate system for the jog keys.

2. Press and hold down the enabling switch.  
Axes A1 to A6 are displayed next to the jog keys.
3. Press the plus or minus jog key to move an axis in the positive or negative direction.
4. Move each axis, starting from A1 and working upwards, so that the mastering marks line up. (An exception is made for A6 of robots for which this axis is mastered using the mark.)

### 5.10.3 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**

- The jog mode "Jog keys" is active.
- Operating mode T1
- No program is selected.
- The user knows the approximate pre-mastering position of the axes.

**NOTICE**

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

**Procedure**

1. Jog the robot to a position in which the axes are close to their pre-mastering position. It should subsequently be possible to move them in the minus direction to the pre-mastering position.
2. In the main menu, select **Start-up > Master > EMD > With load correction**.  
Depending on the method for which the axes are to be aligned, the option **First mastering** or **Teach offset** or **With offset** is now selected.
3. Proceed in accordance with the instructions for the relevant mastering procedure until the probe is attached to A1 and connected via the mastering box to X32.



Thereafter, do NOT continue to follow the description of the mastering procedure!  
i.e. do NOT press **Master** or **Learn** or **Check**!

4. The LED **EMD in mastering range** is displayed on the smartHMI. It must now be red. Observe this LED closely.  
(>>> 5.10.4 "Mastering LEDs" Page 108)
5. Jog the robot in the minus direction. As soon as the LED switches from red to green, stop the robot.  
A1 is now in the pre-mastering position.



The axes indicated next to the LEDs do not disappear one after the other in the usual way. This does not occur until the actual mastering.



Do not yet master the axis. The actual mastering operation must not be carried out until all axes are in the pre-mastering position. If this is not observed, correct mastering cannot be achieved.

6. Remove the probe from the gauge cartridge as described in the mastering procedure and replace the protective cap.

7. Move the remaining axes to the pre-mastering position in the same way in ascending order. (An exception is made for A6 of robots for which this axis is mastered using the mark.)
8. Close the window containing the mastering LEDs.
9. Disconnect the EtherCAT cable from X32 and the mastering box.

**NOTICE**

Leave the signal cable connected to the mastering box and disconnect it as little as possible. The pluggability of the M8 sensor connector is limited. Frequent connection/disconnection can result in damage to the connector.

#### 5.10.4 Mastering LEDs

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



**Fig. 5-5: Mastering LEDs**

LED	Description
<b>Connection to EMD</b>	<ul style="list-style-type: none"> <li>■ <b>Red:</b> The probe is not connected to connection X32.</li> <li>■ <b>Green:</b> The probe is connected to connection X32.</li> </ul> <p>If this LED is red, the LED <b>EMD in mastering range</b> is gray.</p>
<b>EMD in mastering range</b>	<ul style="list-style-type: none"> <li>■ <b>Gray:</b> The probe is not connected to connection X32.</li> <li>■ <b>Red:</b> The probe is in a position where mastering is not possible.</li> <li>■ <b>Green:</b> The probe is either immediately next to or in the mastering notch.</li> </ul>

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.

(>>> 5.10.3 "Moving axes to the pre-mastering position using the probe"  
Page 107)

## 5.10.5 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	<b>First mastering</b> (>>> 5.10.5.1 "First mastering (with SEMD)" Page 109) First mastering is carried out without a load.
2	<b>Teach offset</b> (>>> 5.10.5.2 "Teach offset (with SEMD)" Page 112) "Teach offset" is carried out with a load. The difference from the first mastering is saved.
3	If required: <b>Load mastering with offset</b> (>>> 5.10.5.3 "Check load mastering with offset (with SEMD)" Page 113) "Load mastering with offset" is carried out with a load for which an offset has already been taught. Area of application: <ul style="list-style-type: none"> <li>■ Checking first mastering</li> <li>■ 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.</li> </ul>

### 5.10.5.1 First mastering (with SEMD)

#### Precondition

- 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.
- Operating mode T1

#### Procedure

##### **NOTICE**

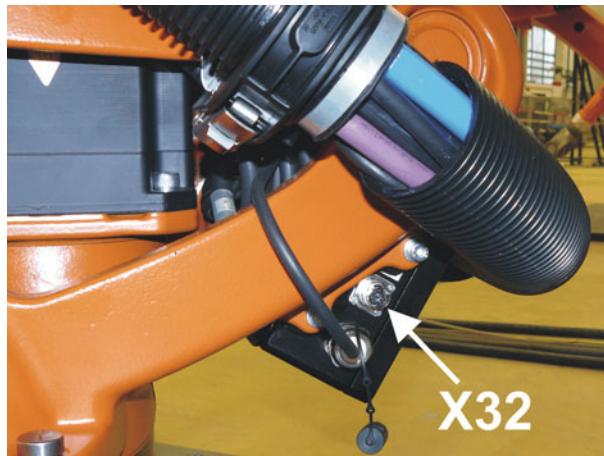
The SEMD must always be screwed onto the gauge cartridge without the signal cable attached. Only then may the cable be attached to the SEMD. Otherwise, the cable could be damaged. Similarly, when removing the SEMD, the signal cable must always be removed from the SEMD first. Only then may the SEMD be removed from the gauge cartridge.

After mastering, remove the EtherCAT cable from connection X32. Failure to do so may result in interference signals or damage.



The SEMD actually used need not necessarily look exactly like the model illustrated in the figures. The procedure for using it is the same, however.

1. In the main menu, select **Start-up > Master > EMD > With load correction > First mastering**.  
 A window opens. All axes to be mastered are displayed. The axis with the lowest number is highlighted.
2. Remove the cover from connection X32.



**Fig. 5-6: Removing cover from X32**

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



**Fig. 5-7: 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. 5-8: Removing protective cap from gauge cartridge**

5. Screw the SEMD onto the gauge cartridge.



**Fig. 5-9: 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. 5-10: Attaching signal cable to SEMD**

7. Connect the signal cable to the mastering box if it is not already connected.
8. Press **Master**.
9. Press an enabling switch and the Start key.  
When the SEMD has passed through the reference notch, the mastering position is calculated. The robot stops automatically. The values are saved. The axis is no longer displayed in the window.
10. Remove the signal cable from the SEMD. Then remove the SEMD from the gauge cartridge and replace the protective cap.
11. Repeat steps 4 to 10 for all axes to be mastered.
12. Close the window.
13. Disconnect the EtherCAT cable from X32 and the mastering box.

**NOTICE**

Leave the signal cable connected to the mastering box and disconnect it as little as possible. The pluggability of the M8 sensor connector is limited. Frequent connection/disconnection can result in damage to the connector.

### 5.10.5.2 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

- 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.
- Operating mode T1

#### Procedure

**NOTICE**

The SEMD must always be screwed onto the gauge cartridge without the signal cable attached. Only then may the cable be attached to the SEMD. Otherwise, the cable could be damaged. Similarly, when removing the SEMD, the signal cable must always be removed from the SEMD first. Only then may the SEMD be removed from the gauge cartridge.

After mastering, remove the EtherCAT cable from connection X32. Failure to do so may result in interference signals or damage.

1. In the main menu, select **Start-up > Master > EMD > With load correction > Teach offset**.
2. Enter tool number. Confirm with **Tool OK**.  
A window opens. All axes for which the tool has not yet been taught are displayed. The axis with the lowest number is highlighted.
3. Remove the cover from connection X32. Connect the EtherCAT cable to X32 and to the mastering box.
4. Remove the protective cap of the gauge cartridge on the axis highlighted in the window. (Turned around, the SEMD can be used as a screwdriver.)
5. Screw the SEMD onto the gauge cartridge.
6. Attach the signal cable to the SEMD. It is possible to see from the cable socket which way round it has to be on the connector pins at the SEMD.
7. Connect the signal cable to the mastering box if it is not already connected.
8. Press **Learn**.
9. Press an enabling switch and the Start key.  
When the SEMD has passed through the reference notch, the mastering position is calculated. The robot stops automatically. A window opens. The deviation of this axis from the first mastering is indicated in degrees and increments.
10. Confirm with **OK**. The axis is no longer displayed in the window.
11. Remove the signal cable from the SEMD. Then remove the SEMD from the gauge cartridge and replace the protective cap.
12. Repeat steps 4 to 11 for all axes to be mastered.
13. Close the window.
14. Disconnect the EtherCAT cable from X32 and the mastering box.

**NOTICE**

Leave the signal cable connected to the mastering box and disconnect it as little as possible. The pluggability of the M8 sensor connector is limited. Frequent connection/disconnection can result in damage to the connector.

### 5.10.5.3 Check load mastering with offset (with SEMD)

<b>Description</b>	<p>Area of application:</p> <ul style="list-style-type: none"> <li>■ Checking first mastering</li> <li>■ 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.</li> </ul> <p>An axis can only be checked if all axes with lower numbers have been mastered.</p>
<b>Precondition</b>	<ul style="list-style-type: none"> <li>■ Same ambient conditions (temperature, etc.) as for first mastering.</li> <li>■ A load for which <b>Teach offset</b> has been carried out is mounted on the robot.</li> <li>■ All axes are in the pre-mastering position.</li> <li>■ No program is selected.</li> <li>■ Operating mode T1</li> </ul>
<b>Procedure</b>	<p><b>NOTICE</b> The SEMD must always be screwed onto the gauge cartridge without the signal cable attached. Only then may the cable be attached to the SEMD. Otherwise, the cable could be damaged. Similarly, when removing the SEMD, the signal cable must always be removed from the SEMD first. Only then may the SEMD be removed from the gauge cartridge.</p> <p>After mastering, remove the EtherCAT cable from connection X32. Failure to do so may result in interference signals or damage.</p> <ol style="list-style-type: none"> <li>1. In the main menu, select <b>Start-up &gt; Master &gt; EMD &gt; With load correction &gt; Load mastering &gt; With offset</b>.</li> <li>2. Enter tool number. Confirm with <b>Tool OK</b>. A window opens. All axes for which an offset has been taught with this tool are displayed. The axis with the lowest number is highlighted.</li> <li>3. Remove the cover from connection X32. Connect the EtherCAT cable to X32 and to the mastering box.</li> <li>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.)</li> <li>5. Screw the SEMD onto the gauge cartridge.</li> <li>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.</li> <li>7. Connect the signal cable to the mastering box if it is not already connected.</li> <li>8. Press <b>Check</b>.</li> <li>9. Hold down an enabling switch and press the Start key. When the SEMD has passed through the reference notch, the mastering position is calculated. The robot stops automatically. The difference from "Teach offset" is displayed.</li> <li>10. If required, press <b>Save</b> to save the values. The old mastering values are deleted. To restore a lost first mastering, always save the values.</li> </ol> <p><b>i</b> Axes A4, A5 and A6 are mechanically coupled. This means: If the values for A4 are deleted, the values for A5 and A6 are also deleted. If the values for A5 are deleted, the values for A6 are also deleted.</p> <ol style="list-style-type: none"> <li>11. Remove the signal cable from the SEMD. Then remove the SEMD from the gauge cartridge and replace the protective cap.</li> </ol>

12. Repeat steps 4 to 11 for all axes to be mastered.
13. Close the window.
14. Disconnect the EtherCAT cable from X32 and the mastering box.

**NOTICE**

Leave the signal cable connected to the mastering box and disconnect it as little as possible. The pluggability of the M8 sensor connector is limited. Frequent connection/disconnection can result in damage to the connector.

### 5.10.6 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. 5-11: Dial gauge

**Precondition**

- The load is mounted on the robot.
- All axes are in the pre-mastering position.
- The jog mode “Jog keys” is active and the coordinate system **Axis** has been selected.
- No program is selected.
- Operating mode T1

**Procedure**

1. In the main menu, select **Start-up > Master > Dial**.

A window opens. All axes that have not been mastered are displayed. The axis that must be mastered first is selected.

2. Remove the protective cap from the gauge cartridge on this axis and mount the dial gauge on the gauge cartridge.

Using the Allen key, loosen the screws on the neck of the dial gauge. Turn the dial so that it can be viewed easily. Push the pin of the dial gauge in as far as the stop.

Using the Allen key, tighten the screws on the neck of the dial gauge.

3. Reduce jog override to 1%.

4. Jog axis from “+” to “-”. At the lowest position of the reference notch, recognizable by the change in direction of the pointer, set the dial gauge to 0. If the axis inadvertently overshoots the lowest position, jog the axis backwards and forwards until the lowest position is reached. It is immaterial whether the axis is moved from “+” to “-” or from “-” to “+”.

5. Move the axis back to the pre-mastering position.

6. Move the axis from “+” to “-” until the pointer is about 5-10 scale divisions before zero.
7. Switch to incremental jogging.
8. Move the axis from “+” to “-” until zero is reached.



If the axis overshoots zero, repeat steps 5 to 8.

9. Press **Master**. The axis that has been mastered is removed from the window.
10. Remove the dial gauge from the gauge cartridge and replace the protective cap.
11. Switch back from incremental jogging to the normal jog mode.
12. Repeat steps 2 to 11 for all axes to be mastered.
13. Close the window.

### 5.10.7 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.

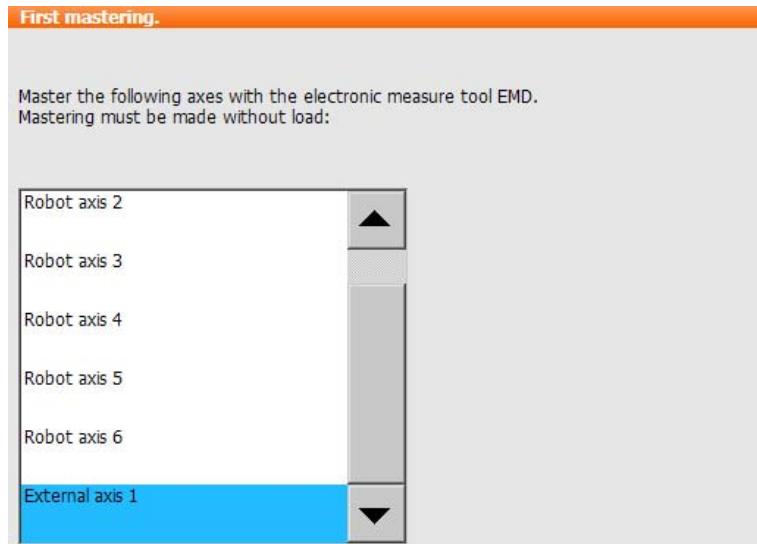


Fig. 5-12: Selection list of axes to be mastered



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.

### 5.10.8 Reference mastering



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

<b>Description</b>	<p>Reference mastering is suitable if maintenance work is due on a correctly mastered robot and it is to be expected that the robot will lose its mastering. Examples:</p> <ul style="list-style-type: none"><li>■ Exchange of RDC</li><li>■ Exchange of motor</li></ul> <p>The robot is moved to the \$MAMES position before the maintenance work is commenced. Afterwards, the axis values of this system variable are reassigned to the robot by means of reference mastering. The state of the robot is then the same as before the loss of mastering. Taught offsets are retained. No EMD or dial gauge is required.</p> <p>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.</p>
<b>Preparation</b>	<ul style="list-style-type: none"><li>■ 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"!</li></ul>
<b>Precondition</b>	<p><b>WARNING</b> 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.</p>
<b>Procedure</b>	<ol style="list-style-type: none"><li>1. In the main menu, select <b>Start-up &gt; Master &gt; Reference</b>. The option window <b>Reference mastering</b> is opened. All axes that have not been mastered are displayed. The axis that must be mastered first is selected.</li><li>2. Press <b>Master</b>. The selected axis is mastered and removed from the option window.</li><li>3. Repeat step 2 for all axes to be mastered.</li></ol>

### 5.10.9 Mastering with the MEMD and mark

<b>Overview</b>	<p>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.</p> <ul style="list-style-type: none"><li>■ In the case of robots with line marks on A6 instead of conventional mastering marks, A6 is mastered without MEMD. (&gt;&gt;&gt; 5.10.9.1 "Moving A6 to the mastering position (with line mark)" Page 117)</li><li>■ In the case of robots with mastering marks on A6, A6 is mastered in the same way as the other axes.</li></ul>
-----------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

<b>Step</b>	<b>Description</b>
1	<b>First mastering</b> (>>> 5.10.9.2 "First mastering (with MEMD)" Page 118) First mastering is carried out without a load.
2	<b>Teach offset</b> (>>> 5.10.9.3 "Teach offset (with MEMD)" Page 121) "Teach offset" is carried out with a load. The difference from the first mastering is saved.
3	If required: <b>Load mastering with offset</b> (>>> 5.10.9.4 "Check load mastering with offset (with MEMD)" Page 122) "Load mastering with offset" is carried out with a load for which an offset has already been taught. Area of application: <ul style="list-style-type: none"><li>■ Checking first mastering</li><li>■ 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.</li></ul>

#### 5.10.9.1 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.

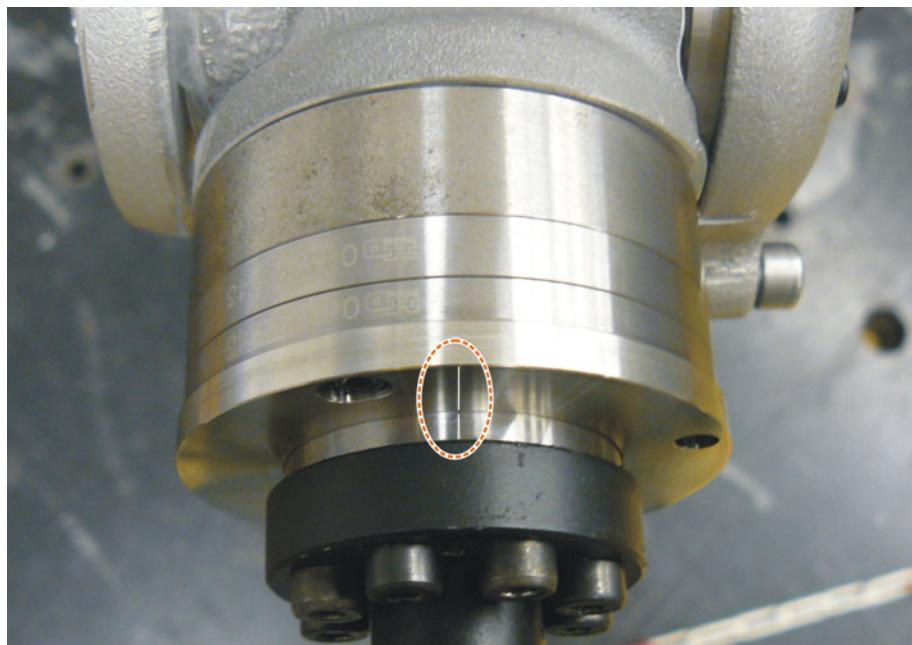


Fig. 5-13: Mastering position A6 – view from above

#### 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.

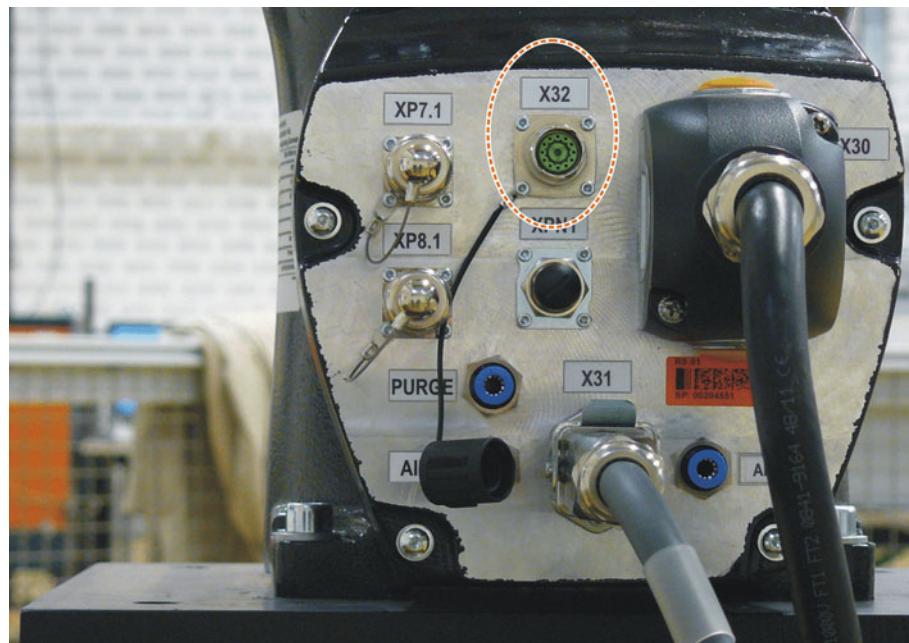
#### 5.10.9.2 First mastering (with MEMD)

##### Precondition

- 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.
- Operating mode T1

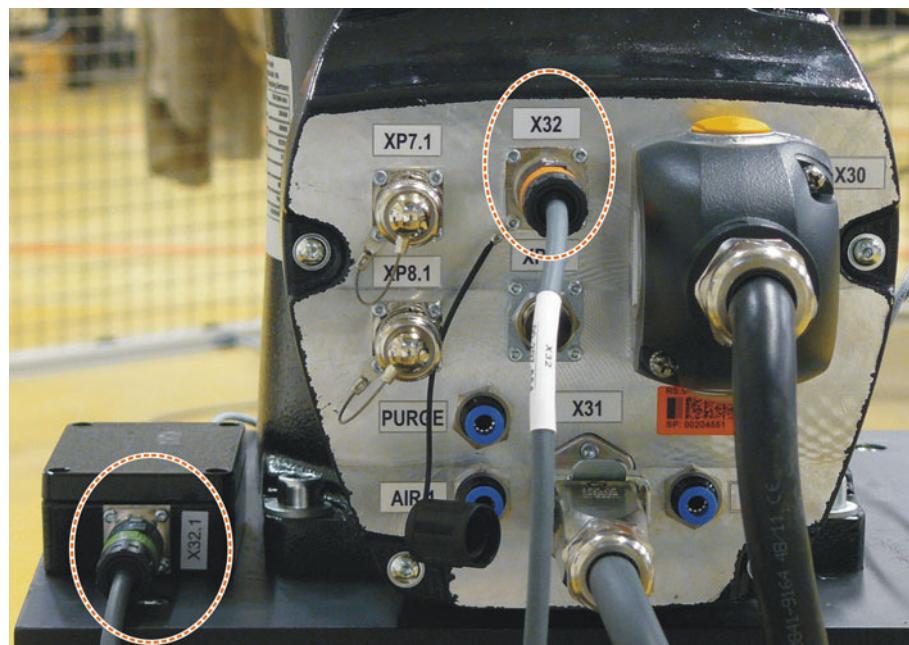
##### Procedure

1. In the main menu, select **Start-up > Master > EMD > With load correction > First mastering**.  
A window opens. All axes to be mastered are displayed. The axis with the lowest number is highlighted.
2. Remove the cover from connection X32.



**Fig. 5-14: X32 without cover**

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



**Fig. 5-15: Connecting the cable to X32**

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



**Fig. 5-16: Removing protective cap from gauge cartridge**

5. Screw the MEMD onto the gauge cartridge.



**Fig. 5-17: Screwing MEMD onto gauge cartridge**

6. Connect the signal cable to the mastering box if it is not already connected.
7. Press **Master**.
8. Press an enabling switch and the Start key.  
When the MEMD has passed through the reference notch, the mastering position is calculated. The robot stops automatically. The values are saved. The axis is no longer displayed in the window.
9. Remove the MEMD from the gauge cartridge and replace the protective cap.
10. Repeat steps 4 to 9 for all axes to be mastered.  
Exception: Not for A6 if this axis has a line mark.
11. Close the window.
12. This step is only to be performed if A6 has a line mark.
  - a. In the main menu, select **Start-up > Master > Reference**.

- The option window **Reference mastering** is opened. A6 is displayed and is selected.
- b. Press **Master**. A6 is mastered and removed from the option window.
  - c. Close the window.
  13. Disconnect the EtherCAT cable from X32 and the mastering box.

**NOTICE**

Leave the signal cable connected to the mastering box and disconnect it as little as possible. The pluggability of the M8 sensor connector is limited. Frequent connection/disconnection can result in damage to the connector.

### 5.10.9.3 Teach offset (with MEMD)

<b>Description</b>	<p><b>Teach offset</b> is carried out with a load. The difference from the first mastering is saved.</p> <p>If the robot is operated with different loads, <b>Teach offset</b> must be carried out for every load. In the case of grippers used for picking up heavy workpieces, <b>Teach offset</b> must be carried out for the gripper both with and without the workpiece.</p>
<b>Precondition</b>	<ul style="list-style-type: none"> <li>■ Same ambient conditions (temperature, etc.) as for first mastering.</li> <li>■ The load is mounted on the robot.</li> <li>■ The axes are in the pre-mastering position. Exception A6, if this axis has a line mark: A6 is in the mastering position.</li> <li>■ No program is selected.</li> <li>■ Operating mode T1</li> </ul>
<b>Procedure</b>	<ol style="list-style-type: none"> <li>1. Select <b>Start-up &gt; Master &gt; EMD &gt; With load correction &gt; Teach offset</b> in the main menu.</li> <li>2. Enter tool number. Confirm with <b>Tool OK</b>. A window opens. All axes for which the tool has not yet been taught are displayed. The axis with the lowest number is highlighted.</li> <li>3. Remove the cover from connection X32.</li> <li>4. Connect the EtherCAT cable to X32 and to the mastering box.</li> <li>5. Remove the protective cap of the gauge cartridge on the axis highlighted in the window.</li> <li>6. Screw the MEMD onto the gauge cartridge.</li> <li>7. Connect the signal cable to the mastering box if it is not already connected.</li> <li>8. Press <b>Learn</b>.</li> <li>9. Press an enabling switch and the Start key. When the MEMD has passed through the reference notch, the mastering position is calculated. The robot stops automatically. A window opens. The deviation of this axis from the first mastering is indicated in degrees and increments.</li> <li>10. Confirm with <b>OK</b>. The axis is no longer displayed in the window.</li> <li>11. Remove the MEMD from the gauge cartridge and replace the protective cap.</li> <li>12. Repeat steps 5 to 11 for all axes to be mastered. Exception: Not for A6 if this axis has a line mark.</li> <li>13. Close the window.</li> <li>14. This step is only to be performed if A6 has a line mark. <ul style="list-style-type: none"> <li>a. In the main menu, select <b>Start-up &gt; Master &gt; Reference</b>.</li> </ul> </li> </ol>

The option window **Reference mastering** is opened. A6 is displayed and is selected.

- b. Press **Master**. A6 is mastered and removed from the option window.
  - c. Close the window.
15. Disconnect the EtherCAT cable from X32 and the mastering box.

**NOTICE**

Leave the signal cable connected to the mastering box and disconnect it as little as possible. The pluggability of the M8 sensor connector is limited. Frequent connection/disconnection can result in damage to the connector.

#### 5.10.9.4 Check load mastering with offset (with MEMD)

<b>Description</b>	<p>Area of application:</p> <ul style="list-style-type: none"><li>■ Checking first mastering</li><li>■ 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.</li></ul> <p>An axis can only be checked if all axes with lower numbers have been mastered.</p> <p>In the case of robots where A6 has a line mark, the value determined for this axis is not displayed, i.e. first mastering cannot be checked for A6. It is possible to restore lost first mastering, however.</p>
<b>Precondition</b>	<ul style="list-style-type: none"><li>■ Same ambient conditions (temperature, etc.) as for first mastering.</li><li>■ A load for which <b>Teach offset</b> has been carried out is mounted on the robot.</li><li>■ The axes are in the pre-mastering position. Exception A6, if this axis has a line mark: A6 is in the mastering position.</li><li>■ No program is selected.</li><li>■ Operating mode T1</li></ul>
<b>Procedure</b>	<ol style="list-style-type: none"><li>1. In the main menu, select <b>Start-up &gt; Master &gt; EMD &gt; With load correction &gt; Master load &gt; With offset</b>.</li><li>2. Enter tool number. Confirm with <b>Tool OK</b>. A window opens. All axes for which an offset has been taught with this tool are displayed. The axis with the lowest number is highlighted.</li><li>3. Remove the cover from connection X32.</li><li>4. Connect the EtherCAT cable to X32 and to the mastering box.</li><li>5. Remove the protective cap of the gauge cartridge on the axis highlighted in the window.</li><li>6. Screw the MEMD onto the gauge cartridge.</li><li>7. Connect the signal cable to the mastering box if it is not already connected.</li><li>8. Press <b>Check</b>.</li><li>9. Hold down an enabling switch and press the Start key. When the MEMD has passed through the reference notch, the mastering position is calculated. The robot stops automatically. The difference from "Teach offset" is displayed.</li><li>10. If required, press <b>Save</b> to save the values. The old mastering values are deleted. To restore a lost first mastering, always save the values.</li></ol>



Axes A4, A5 and A6 are mechanically coupled. This means:  
If the values for A4 are deleted, the values for A5 and A6 are also deleted.

If the values for A5 are deleted, the values for A6 are also deleted.

11. Remove the MEMD from the gauge cartridge and replace the protective cap.
12. Repeat steps 5 to 11 for all axes to be mastered.  
Exception: Not for A6 if this axis has a line mark.
13. Close the window.
14. This step is only to be performed if A6 has a line mark.
  - a. In the main menu, select **Start-up > Master > Reference**.  
The option window **Reference mastering** is opened. A6 is displayed and is selected.
  - b. Press **Master** to restore lost first mastering. A6 is removed from the option window.
  - c. Close the window.
15. Disconnect the EtherCAT cable from X32 and the mastering box.

#### **NOTICE**

Leave the signal cable connected to the mastering box and disconnect it as little as possible. The pluggability of the M8 sensor connector is limited. Frequent connection/disconnection can result in damage to the connector.

### 5.10.10 Manually unmastering axes

#### Description

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



Axes A4, A5 and A6 are mechanically coupled. This means:  
If the values for A4 are deleted, the values for A5 and A6 are also deleted.

If the values for A5 are deleted, the values for A6 are also deleted.

#### **NOTICE**

The software limit switches of an unmastered robot are deactivated. The robot can hit the end stop buffers, thus damaging the robot and making it necessary to exchange the buffers. An unmastered robot must not be jogged, if at all avoidable. If it must be jogged, the jog override must be reduced as far as possible.

#### Precondition

- No program is selected.
- Operating mode T1

#### 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.

### 5.11 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**

- “Expert” user group
- 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:**

Software limit switch			
Axis	Negative	Current position	Positive
A1 [°]	-185.00	0.00	185.00
A2 [°]	-146.00	0.00	0.00
A3 [°]	-119.00	0.00	155.00
A4 [°]	-350.00	0.00	350.00
A5 [°]	-125.00	0.00	125.00
A6 [°]	-350.00	0.00	350.00

**Fig. 5-18: Before automatic determination**

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

Software limit switch			
Axis	Minimum	Current position	Maximum
A1 [°]	0.00	0.00	0.00
A2 [°]	0.00	0.00	0.00
A3 [°]	0.00	0.00	0.00
A4 [°]	0.00	0.00	0.00
A5 [°]	0.00	0.00	0.00
A6 [°]	0.00	0.00	0.00

**Fig. 5-19: 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 (only in the “Expert” user group):

Button	Description
<b>Auto detection</b>	Starts the automatic determination: The robot controller writes the minimum and maximum positions adopted by the axes from now on to the columns <b>Minimum</b> and <b>Maximum</b> in the <b>Software limit switch</b> window.
<b>End</b>	Ends the automatic determination. Transfers the calculated minimum/maximum positions to the columns <b>Negative</b> and <b>Positive</b> , but does not yet save them.
<b>Save</b>	Saves the values in the columns <b>Negative</b> and <b>Positive</b> as software limit switches.

## 5.12 Calibration

### 5.12.1 Defining the tool direction

- Description** By default, the X axis is defined in the system as the tool direction. The tool direction can be changed using the system variable \$TOOL\_DIRECTION.
- The change relates only to spline motions. For LIN and CIRC motions, the tool direction is the X axis and cannot be changed.
  - The change applies to all tools. It is not possible to define different tool directions for different tools.



**WARNING** The tool direction must be defined before calibration and before program creation. It cannot be modified subsequently. Failure to observe this may result in unexpected changes to the motion characteristics of the robot. Death, injuries or damage to property may result.

- Precondition**
- “Expert” user group

- Procedure**
- Set the system variable \$TOOL\_DIRECTION to the desired value in the file \$CUSTOM.DAT, located in the directory KRC\Steu\MaDa.  
Possible values: #X (default); #Y; #Z

It is not possible to modify \$TOOL\_DIRECTION by means of the variable correction function or by writing to the variable from the program.

### 5.12.2 Tool calibration

- Description** During tool calibration, the user assigns a Cartesian coordinate system to the tool mounted 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.



In the case of a fixed tool, the type of calibration described here must not be used. A separate type of calibration must be used for fixed tools. ([>>> 5.12.4 "Fixed tool calibration" Page 136](#))

Advantages of tool calibration:

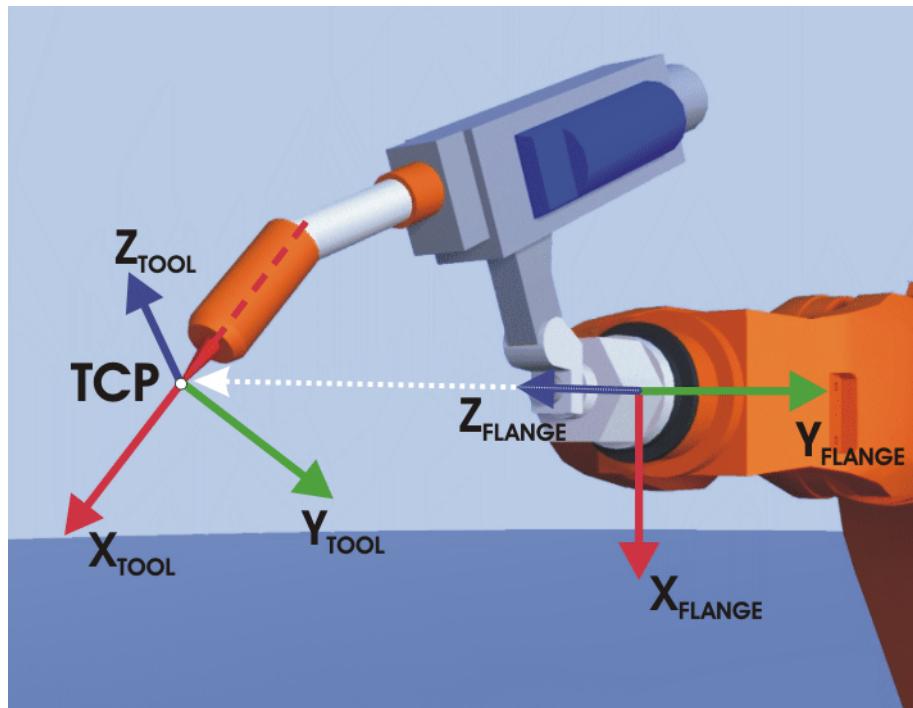
- The tool can be moved in a straight line in the tool direction.

- The tool 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. 5-20: TCP calibration principle**

## Overview

Tool calibration consists of 2 steps:

Step	Description
1	<b>Definition of the origin of the TOOL coordinate system</b> The following methods are available: <ul style="list-style-type: none"> <li>■ XYZ 4-point</li> <li>■ XYZ Reference</li> </ul>
2	<b>Definition of the orientation of the TOOL coordinate system</b> The following methods are available: <ul style="list-style-type: none"> <li>■ ABC 2-point</li> <li>■ ABC World</li> </ul>

If the calibration data are already known, they can be entered directly.  
 (>>> 5.12.2.5 "Numeric input" Page 132)

### 5.12.2.1 TCP calibration: XYZ 4-point method



The XYZ 4-point method cannot be used for palletizing robots.

#### 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 4 flange positions at the reference point must be sufficiently different from one another and must not lie in a plane.

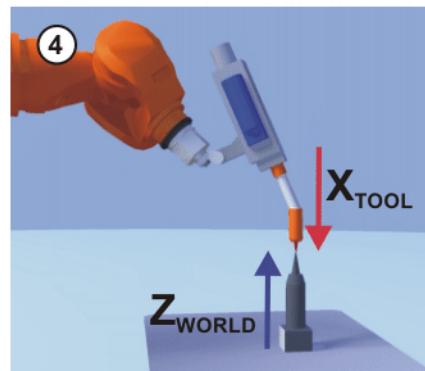
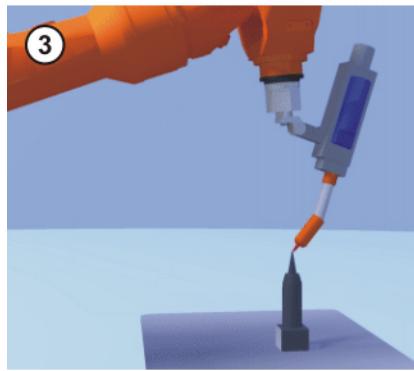
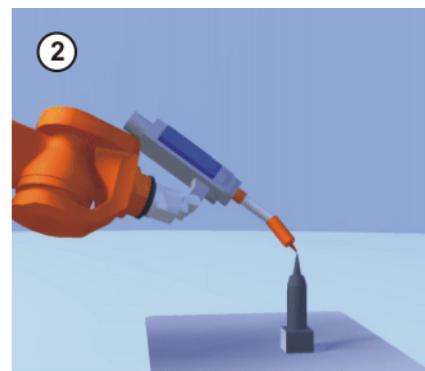
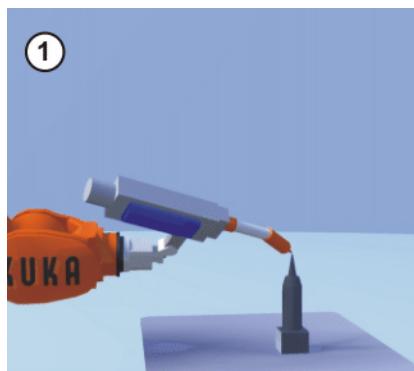


Fig. 5-21: XYZ 4-point method

#### Precondition

- The tool to be calibrated is mounted on the mounting flange.
- Operating mode T1

#### Procedure

1. In the main menu, select **Start-up > Calibrate > Tool > XYZ 4-point**.
2. Select a number for the tool that is to be calibrated and assign a tool name. Confirm with **Next**.
3. Move the TCP to a reference point. Press **Calibrate**. Answer the request for confirmation with **Yes**.
4. Move the TCP to the reference point from a different direction. Press **Calibrate**. Answer the request for confirmation with **Yes**.
5. Repeat step 4 twice.
6. Enter the payload data. (This step can be skipped if the payload data are entered separately instead.)  
*(>>> 5.13.3 "Entering payload data" Page 150)*
7. Confirm with **Next**.

8. If required, coordinates and orientation of the calibrated points can be displayed in increments and degrees (relative to the FLANGE coordinate system). For this, press **Meas. points**. Then return to the previous view by pressing **Back**.
9. Either: Press **Save** and then close the window via the **Close** icon.  
Or: Press **ABC 2-point** or **ABC World**. The previous data are automatically saved and a window is opened in which the orientation of the TOOL coordinate system can be defined.  
(>>> 5.12.2.4 "Defining the orientation: ABC 2-point method" Page 131)  
(>>> 5.12.2.3 "Defining the orientation: ABC World method" Page 130)

### 5.12.2.2 TCP calibration: XYZ Reference method

<b>Description</b>	In the case of the XYZ Reference method, a new tool is calibrated with a tool that has already been calibrated. The robot controller compares the flange positions and calculates the TCP of the new tool.
--------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

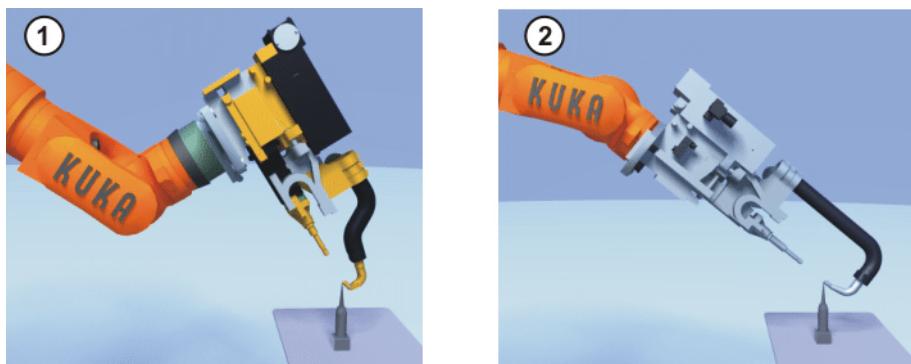


Fig. 5-22: XYZ Reference method

<b>Precondition</b>	<ul style="list-style-type: none"> <li>■ A previously calibrated tool is mounted on the mounting flange.</li> <li>■ Operating mode T1</li> </ul>
<b>Preparation</b>	<p>Calculate the TCP data of the calibrated tool:</p> <ol style="list-style-type: none"> <li>1. In the main menu, select <b>Start-up &gt; Calibrate &gt; Tool &gt; XYZ Reference</b>.</li> <li>2. Select the number of the calibrated tool.</li> <li>3. The tool data are displayed. Note the X, Y and Z values.</li> <li>4. Close the window.</li> </ol>
<b>Procedure</b>	<ol style="list-style-type: none"> <li>1. In the main menu, select <b>Start-up &gt; Calibrate &gt; Tool &gt; XYZ Reference</b>.</li> <li>2. Select a number for the new tool and assign a tool name. Confirm with <b>Next</b>.</li> <li>3. Enter the TCP data of the calibrated tool. Confirm with <b>Next</b>.</li> <li>4. Move the TCP to a reference point. Press <b>Calibrate</b>. Answer the request for confirmation with <b>Yes</b>.</li> <li>5. Move the tool away and remove it. Mount the new tool.</li> <li>6. Move the TCP of the new tool to the reference point. Press <b>Calibrate</b>. Answer the request for confirmation with <b>Yes</b>.</li> <li>7. Enter the payload data. (This step can be skipped if the payload data are entered separately instead.) (&gt;&gt;&gt; 5.13.3 "Entering payload data" Page 150)</li> <li>8. Confirm with <b>Next</b>.</li> <li>9. If required, coordinates and orientation of the calibrated points can be displayed in increments and degrees (relative to the FLANGE coordinate sys-</li> </ol>

tem). For this, press **Meas. points**. Then return to the previous view by pressing **Back**.

10. Either: Press **Save** and then close the window via the **Close** icon.

Or: Press **ABC 2-point** or **ABC World**. The previous data are automatically saved and a window is opened in which the orientation of the TOOL coordinate system can be defined.

(>>> 5.12.2.4 "Defining the orientation: ABC 2-point method" Page 131)

(>>> 5.12.2.3 "Defining the orientation: ABC World method" Page 130)

### 5.12.2.3 Defining the orientation: ABC World method

#### Description

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.

There are 2 variants of this method:

- **5D**: The user communicates the tool direction to the robot controller. By default, the tool direction is the X axis. The orientation of the other axes is defined by the system and cannot be influenced by the user.

The system always defines the orientation of the other axes in the same way. If the tool subsequently has to be calibrated again, e.g. after a crash, it is therefore sufficient to define the tool direction again. Rotation about the tool direction need not be taken into consideration.

- **6D**: The user communicates the direction of all 3 axes to the robot controller.

#### Precondition

- The tool to be calibrated is mounted on the mounting flange.
- The TCP of the tool has already been measured.
- T1 mode



The following procedure applies if the tool direction is the default tool direction (= X axis). If the tool direction has been changed to Y or Z, the procedure must also be changed accordingly.

(>>> 5.12.1 "Defining the tool direction" Page 126)

#### Procedure

1. In the main menu, select **Start-up > Calibrate > Tool > ABC World**.
2. Select the number of the tool to be calibrated. Confirm with **Next**.
3. Select a variant in the **5D / 6D** box. Confirm with **Next**.
4. If **5D** is selected:

Align  $+X_{TOOL}$  parallel to  $-Z_{WORLD}$ . ( $+X_{TOOL}$  = tool direction)

If **6D** is selected:

Align the axes of the TOOL coordinate system as follows.

- $+X_{TOOL}$  parallel to  $-Z_{WORLD}$ . ( $+X_{TOOL}$  = tool direction)
- $+Y_{TOOL}$  parallel to  $+Y_{WORLD}$ .
- $+Z_{TOOL}$  parallel to  $+X_{WORLD}$ .

5. Press **Calibrate**. Answer the request for confirmation with **Yes**.



The following two steps are not required if the procedure is not called via the main menu, but by means of the **ABC World** button after TCP calibration.

6. Enter the payload data. (This step can be skipped if the payload data are entered separately instead.)  
(>>> 5.13.3 "Entering payload data" Page 150)
7. Confirm with **Next**.

8. If required, coordinates and orientation of the calibrated points can be displayed in increments and degrees (relative to the FLANGE coordinate system). For this, press **Meas. points**. Then return to the previous view by pressing **Back**.
9. Press **Save**.

#### 5.12.2.4 Defining the orientation: ABC 2-point method

##### Description

The axes of the TOOL coordinate system are communicated to the robot controller by moving to a point on the X axis and a point in the XY plane.

This method is used if it is necessary to define the axis directions with particular precision.

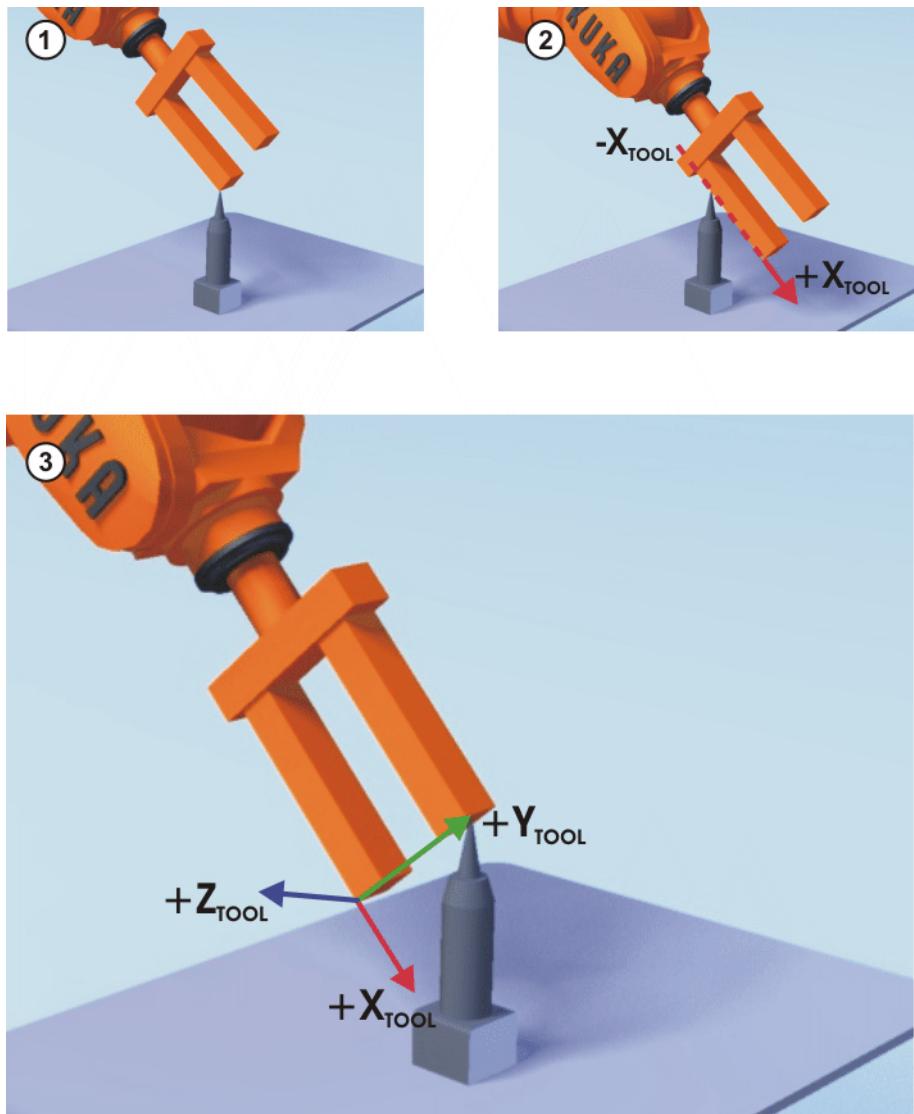


Fig. 5-23: ABC 2-point method

##### Precondition

- The tool to be calibrated is mounted on the mounting flange.
- The TCP of the tool has already been measured.
- T1 mode



The following procedure applies if the tool direction is the default tool direction (= X axis). If the tool direction has been changed to Y or Z, the procedure must also be changed accordingly.

(>>> 5.12.1 "Defining the tool direction" Page 126)

- |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Procedure</b> | <ol style="list-style-type: none"><li>1. In the main menu, select <b>Start-up &gt; Calibrate &gt; Tool &gt; ABC 2-point</b>.</li><li>2. Select the number of the tool to be calibrated. Confirm with <b>Next</b>.</li><li>3. Move the TCP to any reference point. Press <b>Calibrate</b>. Answer the request for confirmation with <b>Yes</b>.</li><li>4. Move the tool so that the reference point on the X axis has a negative X value (i.e. move against the tool direction). Press <b>Calibrate</b>. Answer the request for confirmation with <b>Yes</b>.</li><li>5. Move the tool so that the reference point in the XY plane has a positive Y value. Press <b>Calibrate</b>. Answer the request for confirmation with <b>Yes</b>.</li></ol> |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
-  The following two steps are not required if the procedure is not called via the main menu, but by means of the **ABC 2-point** button after TCP calibration.
6. Enter the payload data. (This step can be skipped if the payload data are entered separately instead.)  
(>>> 5.13.3 "Entering payload data" Page 150)
  7. Confirm with **Next**.
  8. If required, coordinates and orientation of the calibrated points can be displayed in increments and degrees (relative to the FLANGE coordinate system). For this, press **Meas. points**. Then return to the previous view by pressing **Back**.
  9. Press **Save**.

#### 5.12.2.5 Numeric input

- |                    |                                                                                                                                                                                                                               |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | <p>The tool data can be entered manually.</p> <p>Possible sources of data:</p> <ul style="list-style-type: none"><li>■ CAD</li><li>■ Externally calibrated tool</li><li>■ Specifications from the tool manufacturer</li></ul> |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
-  In the case of palletizing robots with 4 axes, the tool data must be entered numerically. The different XYZ and ABC methods cannot be used, as reorientation of these robots is highly restricted.
- |                     |                                                                                                                                                                                                                                                                                  |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Precondition</b> | <ul style="list-style-type: none"><li>■ The following values are known:<ul style="list-style-type: none"><li>■ X, Y and Z relative to the FLANGE coordinate system</li><li>■ A, B and C relative to the FLANGE coordinate system</li></ul></li><li>■ Operating mode T1</li></ul> |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
- |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Procedure</b> | <ol style="list-style-type: none"><li>1. In the main menu, select <b>Start-up &gt; Calibrate &gt; Tool &gt; Numeric input</b>.</li><li>2. Select a number for the tool and assign a tool name. Confirm with <b>Next</b>.</li><li>3. Enter the tool data. Confirm with <b>Next</b>.</li><li>4. Enter the payload data. (This step can be skipped if the payload data are entered separately instead.)<br/>(&gt;&gt;&gt; 5.13.3 "Entering payload data" Page 150)</li><li>5. If online load data checking is available (this depends on the robot type): configure as required.<br/>(&gt;&gt;&gt; 5.13.5 "Online load data check (OLDC)" Page 151)</li><li>6. Confirm with <b>Next</b>.</li><li>7. Press <b>Save</b>.</li></ol> |
|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### 5.12.3 Base calibration

#### Description

During base calibration, the user assigns a Cartesian coordinate system (BASE coordinate system) to a work surface or the workpiece. The BASE coordinate system has its origin at a user-defined point.



If the workpiece is mounted on the mounting flange, the type of calibration described here must not be used. A separate type of calibration must be used for workpieces mounted on the mounting flange.  
(>>> 5.12.4 "Fixed tool calibration" Page 136)

Advantages of base calibration:

- The TCP can be jogged along the edges of the work surface or workpiece.
- 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].

#### Overview

There are 2 ways of calibrating a base:

- 3-point method  
(>>> 5.12.3.1 "Base calibration: 3-point method" Page 133)
- Indirect method  
(>>> 5.12.3.2 "Base calibration: indirect method" Page 135)

If the calibration data are already known, they can be entered directly.

(>>> 5.12.3.3 "Entering the base numerically" Page 136)

#### 5.12.3.1 Base calibration: 3-point method

#### Description

The robot moves to the origin and 2 further points of the new base. These 3 points define the new base.

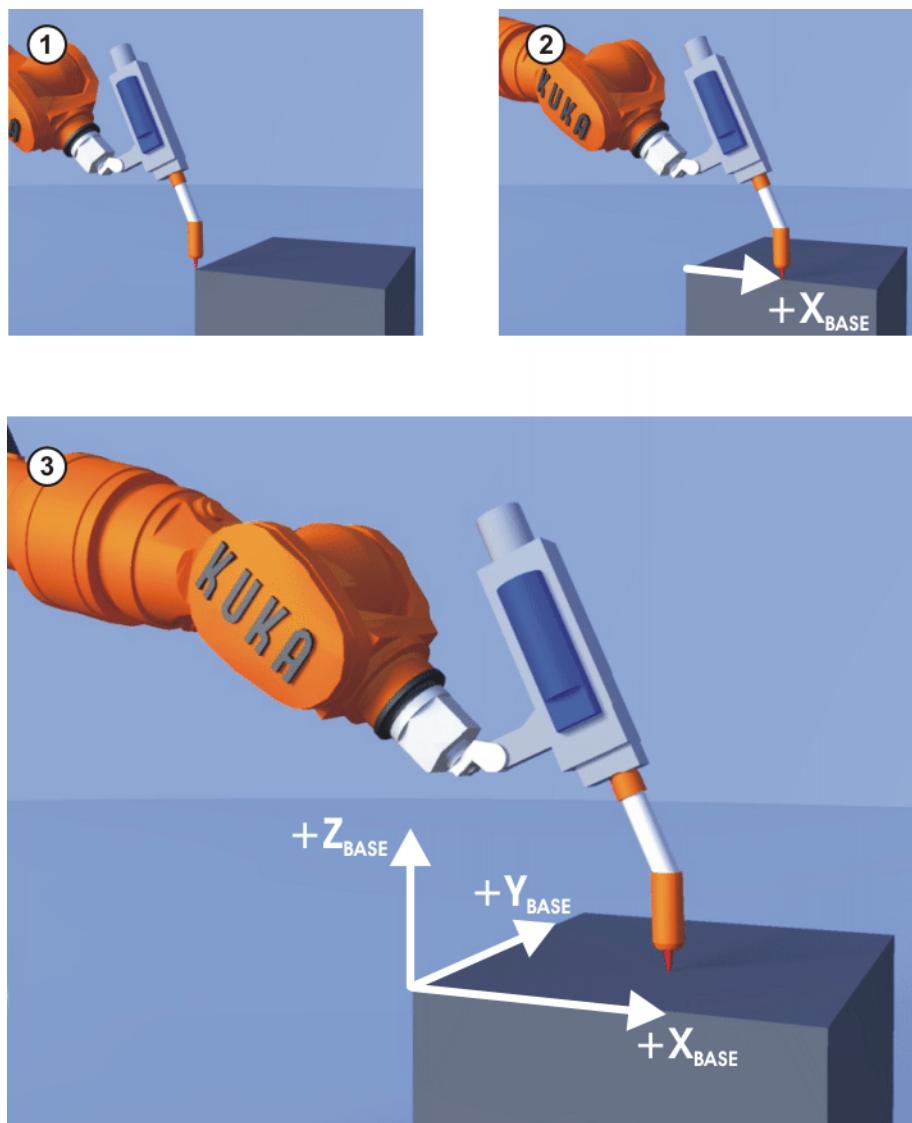


Fig. 5-24: 3-point method

**Precondition**

- A previously calibrated tool is mounted on the mounting flange.
- Operating mode T1

**Procedure**

1. In the main menu, select **Start-up > Calibrate > Base > 3-point**.
2. Assign a number and a name for the base that is to be calibrated. Confirm with **Next**.
3. Select the number of the tool that has already been calibrated. Confirm with **Next**.
4. Move the TCP to the origin of the new base. Press **Calibrate**. Answer the request for confirmation with **Yes**.
5. Move the TCP to a point on the positive X axis of the new base. Press **Calibrate**. Answer the request for confirmation with **Yes**.
6. Move the TCP to a point in the XY plane with a positive Y value. Press **Calibrate**. Answer the request for confirmation with **Yes**.
7. If required, coordinates and orientation of the calibrated points can be displayed in increments and degrees (relative to the FLANGE coordinate system). For this, press **Meas. points**. Then return to the previous view by pressing **Back**.
8. Press **Save**.

### 5.12.3.2 Base calibration: indirect method

#### Description

The indirect method is used if it is not possible to move to the origin of the base, e.g. because it is inside a workpiece or outside the workspace of the robot.

The TCP is moved to 4 points in the base, the coordinates of which must be known. The robot controller calculates the base from these points.

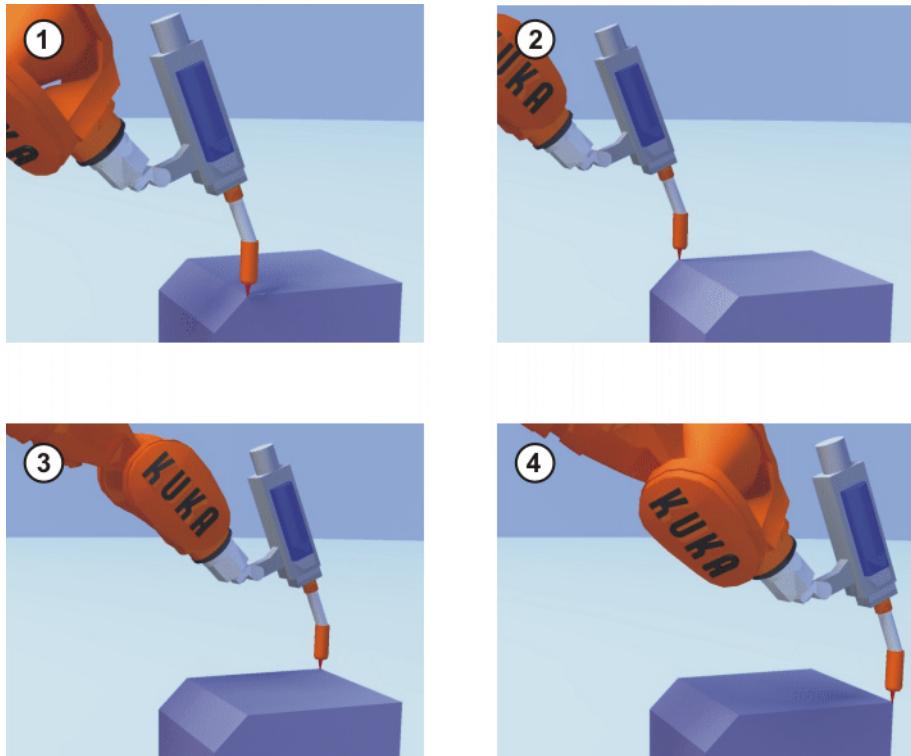


Fig. 5-25: Indirect method

#### Precondition

- A previously calibrated tool is mounted on the mounting 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.
- Operating mode T1

#### Procedure

1. In the main menu, select **Start-up > Calibrate > Base > Indirect**.
2. Assign a number and a name for the base that is to be calibrated. Confirm with **Next**.
3. Select the number of the tool that has already been calibrated. Confirm with **Next**.
4. Enter the coordinates of a known point in the new base and move the TCP to this point. Press **Calibrate**. Answer the request for confirmation with **Yes**.
5. Repeat step 4 three times.
6. If required, coordinates and orientation of the calibrated points can be displayed in increments and degrees (relative to the FLANGE coordinate system). For this, press **Meas. points**. Then return to the previous view by pressing **Back**.
7. Press **Save**.

### 5.12.3.3 Entering the base numerically

- Precondition**
- The following numerical values are known, e.g. from CAD data:
    - Distance between the origin of the base and the origin of the WORLD coordinate system
    - Rotation of the base axes relative to the WORLD coordinate system
  - Operating mode T1
- Procedure**
1. In the main menu, select **Start-up > Calibrate > Base > Numeric input.**
  2. Assign a number and a name for the base that is to be calibrated. Confirm with **Next**.
  3. Enter data. Confirm with **Next**.
  4. Press **Save**.

### 5.12.4 Fixed tool calibration

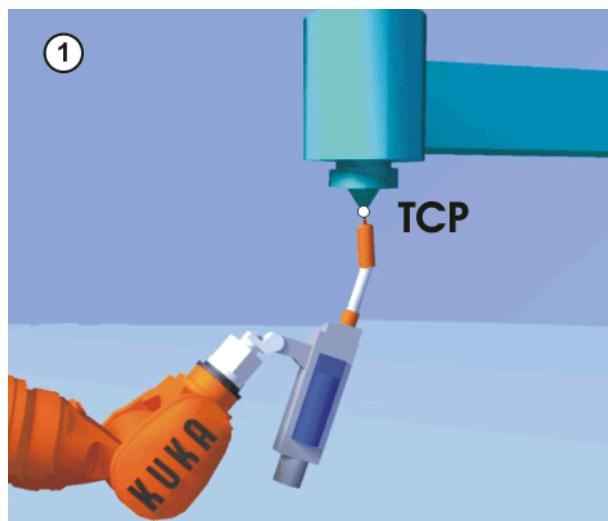
**Overview** Calibration of a fixed tool consists of 2 steps:

Step	Description
1	<p><b>Calibration of the TCP of the fixed tool</b></p> <p>The TCP of a fixed tool is called an external TCP.</p> <p>(&gt;&gt;&gt; 5.12.4.1 "Calibrating an external TCP" Page 136)</p> <p>If the calibration data are already known, they can be entered directly.</p> <p>(&gt;&gt;&gt; 5.12.4.2 "Entering the external TCP numerically" Page 138)</p>
2	<p><b>Calibration of the workpiece</b></p> <p>The following methods are available:</p> <ul style="list-style-type: none"> <li>■ Direct method</li> <li>■ Indirect method</li> </ul>

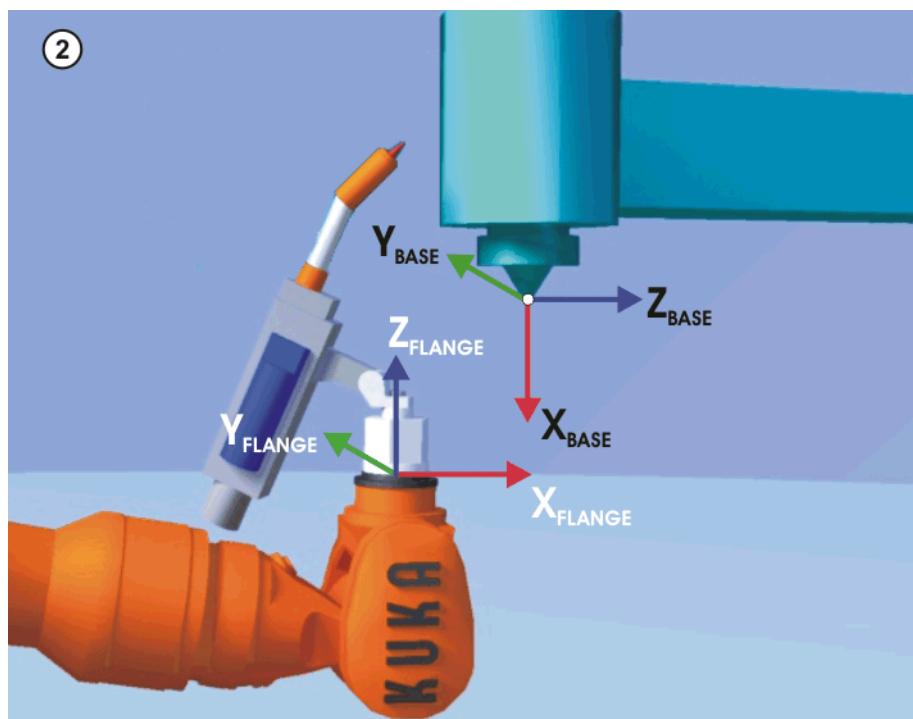
The robot controller saves the external TCP as the BASE coordinate system and the workpiece as the TOOL coordinate system.

#### 5.12.4.1 Calibrating an external TCP

- Description**
- First of all, the TCP of the fixed tool is communicated to the robot controller. This is done by moving a calibrated tool to it.
- Then, the orientation of the coordinate system of the fixed tool is communicated to the robot controller. For this purpose, the coordinate system of the calibrated tool is aligned parallel to the new coordinate system. There are 2 variants:
- **5D:** Only the tool direction of the fixed tool is communicated to the robot controller. By default, the tool direction is the X axis. The orientation of the other axes is defined by the system and cannot be detected easily by the user.
  - **6D:** The orientation of all 3 axes is communicated to the robot controller.



**Fig. 5-26: Moving to the external TCP**



**Fig. 5-27: Aligning the coordinate systems parallel to one another**

#### Precondition

- A previously calibrated tool is mounted on the mounting flange.
- Operating mode T1



The following procedure applies if the tool direction is the default tool direction (= X axis). If the tool direction has been changed to Y or Z, the procedure must also be changed accordingly.

(>>> 5.12.1 "Defining the tool direction" Page 126)

#### Procedure

1. In the main menu, select **Start-up > Calibrate > Fixed tool > Tool**.
2. Select a number for the fixed tool that is to be calibrated and assign a tool name. Confirm with **Next**.
3. Select the number of the tool that has already been calibrated. Confirm with **Next**.
4. Select a variant in the **5D / 6D** box. Confirm with **Next**.

5. Move the TCP of the calibrated tool to the TCP of the fixed tool. Press **Calibrate**. Answer the request for confirmation with **Yes**.
6. If **5D** is selected:  
Align  $+X_{BASE}$  parallel to  $-Z_{FLANGE}$   
(i.e. align the mounting flange perpendicular to the tool direction of the fixed tool.)  
If **6D** is selected:  
Align the mounting flange so that its axes are parallel to the axes of the fixed tool:
  - $+X_{BASE}$  parallel to  $-Z_{FLANGE}$   
(i.e. align the mounting flange perpendicular to the tool direction.)
  - $+Y_{BASE}$  parallel to  $+Y_{FLANGE}$
  - $+Z_{BASE}$  parallel to  $+X_{FLANGE}$
7. Press **Calibrate**. Answer the request for confirmation with **Yes**.
8. If required, coordinates and orientation of the calibrated points can be displayed in increments and degrees (relative to the FLANGE coordinate system). For this, press **Meas. points**. Then return to the previous view by pressing **Back**.
9. Press **Save**.

#### 5.12.4.2 Entering the external TCP numerically

##### Precondition

- The following numerical values are known, e.g. from CAD data:
  - Distance between the TCP of the fixed tool and the origin of the WORLD coordinate system (X, Y, Z)
  - Rotation of the axes of the fixed tool relative to the WORLD coordinate system (A, B, C)
- Operating mode T1

##### Procedure

1. In the main menu, select **Start-up > Calibrate > Fixed tool > Numeric input**.
2. Select a number for the fixed tool and assign a tool name. Confirm with **Next**.
3. Enter data. Confirm with **Next**.
4. Press **Save**.

#### 5.12.4.3 Workpiece calibration: direct method

##### Description

The origin and 2 further points of the workpiece are communicated to the robot controller. These 3 points uniquely define the workpiece.

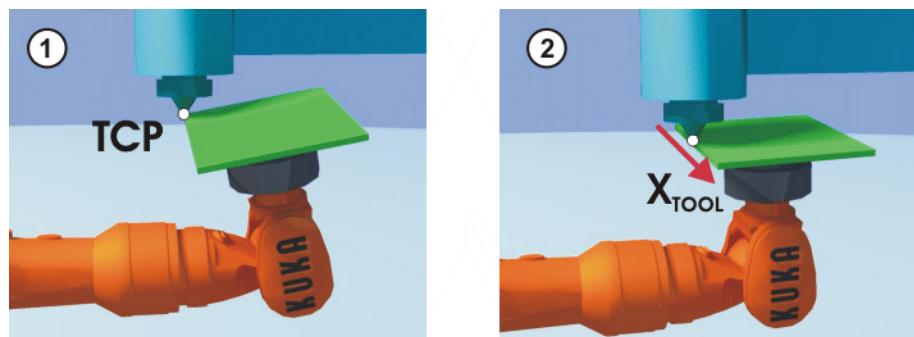
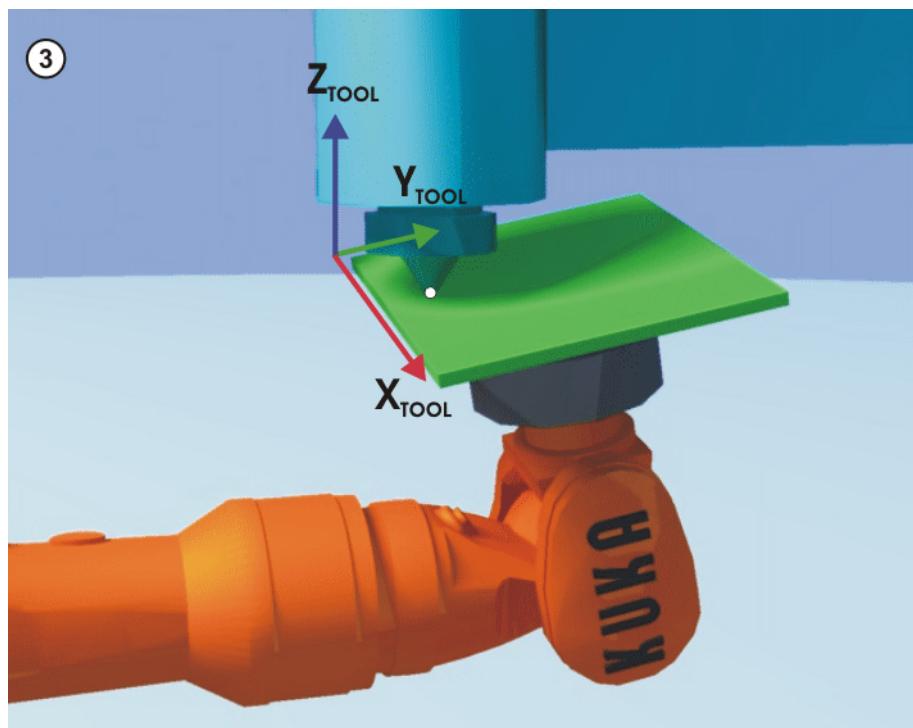


Fig. 5-28



**Fig. 5-29: Workpiece calibration: direct method**

#### Precondition

- The workpiece to be calibrated is mounted on the mounting flange.
- A previously calibrated fixed tool is mounted.
- Operating mode T1

#### Procedure

1. In the main menu, select **Start-up > Calibrate > Fixed tool > Workpiece > Direct calibration**.
2. Select a number for the workpiece to be calibrated and assign a workpiece name. Confirm with **Next**.
3. Select the number of the fixed tool. Confirm with **Next**.
4. Move the origin of the workpiece coordinate system to the TCP of the fixed tool. Press **Calibrate**. Answer the request for confirmation with **Yes**.
5. Move a point on the positive X axis of the workpiece coordinate system to the TCP of the fixed tool. Press **Calibrate**. Answer the request for confirmation with **Yes**.
6. Move a point with a positive Y value in the XY plane of the workpiece coordinate system to the TCP of the fixed tool. Press **Calibrate**. Answer the request for confirmation with **Yes**.
7. Enter the load data of the workpiece. (This step can be skipped if the load data are entered separately instead.)  
(>>> 5.13.3 "Entering payload data" Page 150)
8. Confirm with **Next**.
9. If required, coordinates and orientation of the calibrated points can be displayed in increments and degrees (relative to the FLANGE coordinate system). For this, press **Meas. points**. Then return to the previous view by pressing **Back**.
10. Press **Save**.

#### 5.12.4.4 Workpiece calibration: indirect method

**Description**

The robot controller calculates the workpiece on the basis of 4 points whose coordinates must be known. The robot does not move to the origin of the workpiece.

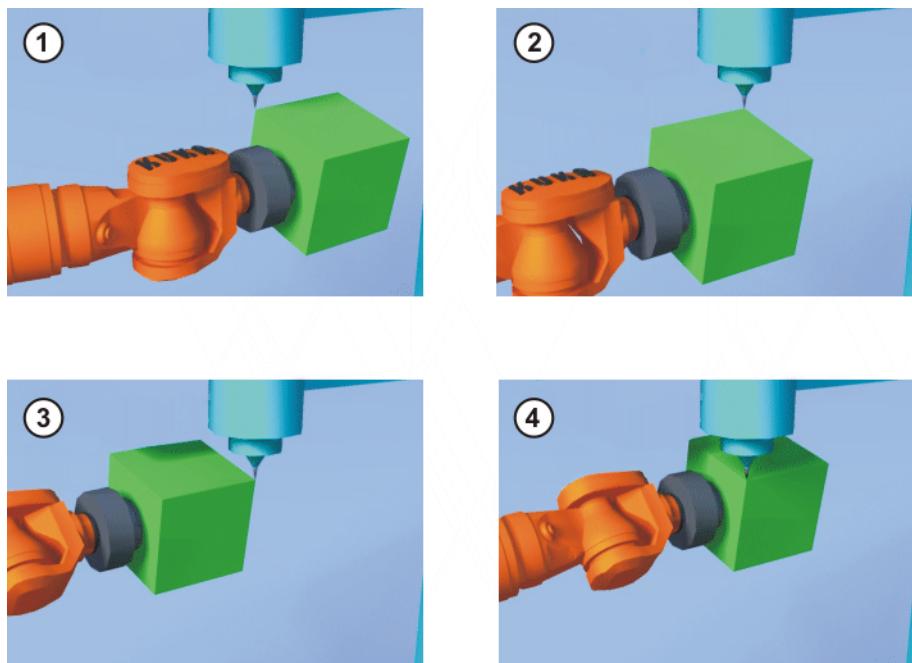


Fig. 5-30: Workpiece calibration: indirect method

**Precondition**

- The workpiece to be calibrated is mounted on the mounting flange.
- The coordinates of 4 points of the new workpiece are known, e.g. from CAD data. The 4 points are accessible to the TCP.
- A previously calibrated fixed tool is mounted.
- Operating mode T1

**Procedure**

1. In the main menu, select **Start-up > Calibrate > Fixed tool > Workpiece > Indirect calibration**.
2. Select a number for the workpiece to be calibrated and assign a workpiece name. Confirm with **Next**.
3. Select the number of the fixed tool. Confirm with **Next**.
4. Enter the coordinates of a known point on the workpiece and move this point to the TCP of the fixed tool. Press **Calibrate**. Answer the request for confirmation with **Yes**.
5. Repeat step 4 three times.
6. Enter the load data of the workpiece. (This step can be skipped if the load data are entered separately instead.)  
(>>> 5.13.3 "Entering payload data" Page 150)
7. Confirm with **Next**.
8. If required, coordinates and orientation of the calibrated points can be displayed in increments and degrees (relative to the FLANGE coordinate system). For this, press **Meas. points**. Then return to the previous view by pressing **Back**.
9. Press **Save**.

#### 5.12.5 Renaming the tool/base

**Precondition**

- Operating mode T1

- |                  |                                                                                                                                                                                                                                                                          |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Procedure</b> | <ol style="list-style-type: none"> <li>1. In the main menu, select <b>Start-up &gt; Calibrate &gt; Tool or Base &gt; Change name.</b></li> <li>2. Select the tool or base and press <b>Name</b>.</li> <li>3. Enter the new name and confirm with <b>Save</b>.</li> </ol> |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### 5.12.6 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 \$ERSYS-ROOT

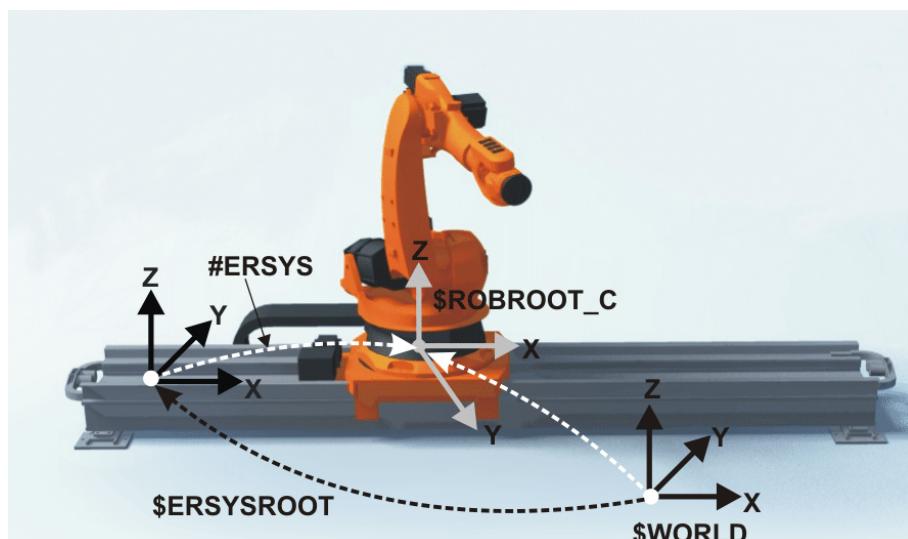


Fig. 5-31: ROBROOT kinematic system – linear unit

#### 5.12.6.1 Checking whether the linear unit needs to be calibrated

- |                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | <p>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.)</p> |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

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.

<b>Precondition</b>	<ul style="list-style-type: none"><li>■ The machine data of the linear unit have been configured and loaded into the robot controller.</li><li>■ A previously calibrated tool is mounted on the mounting flange.</li><li>■ No program is open or selected.</li><li>■ Operating mode T1</li></ul>
<b>Procedure</b>	<ol style="list-style-type: none"><li>1. Align the TCP against a freely selected point and observe it.</li><li>2. Execute a Cartesian (not axis-specific) motion with the linear unit.<ul style="list-style-type: none"><li>■ If the TCP remains stationary: the linear unit does not require calibration.</li><li>■ If the TCP moves: the linear unit does require calibration.</li></ul></li></ol> <p>If the calibration data are already known (e.g. from CAD), they can be entered numerically.</p>

### 5.12.6.2 Calibrating the linear unit

<b>Description</b>	<p>During calibration, the TCP of a tool that has already been calibrated is moved to a reference point 3 times.</p> <ul style="list-style-type: none"><li>■ The reference point can be freely selected.</li><li>■ 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.</li></ul> <p>The correction values determined by the calibration are factored into the system variable \$ETx_TFLA3.</p>
<b>Precondition</b>	<ul style="list-style-type: none"><li>■ The machine data of the linear unit have been configured and loaded into the robot controller.</li><li>■ A previously calibrated tool is mounted on the mounting flange.</li><li>■ No program is open or selected.</li><li>■ Operating mode T1</li></ul>
<b>Procedure</b>	<ol style="list-style-type: none"><li>1. In the main menu, select <b>Start-up &gt; Calibrate &gt; External kinematic system &gt; Linear unit</b>.</li><li>2. Select the number of the tool that has already been calibrated. Confirm with <b>Next</b>.</li></ol> <p>The robot controller detects the linear unit automatically and displays the following data:</p> <ul style="list-style-type: none"><li>■ <b>Ext. kinematic system no.:</b> number of the external kinematic system (1 ... 6) (\$EX_KIN)</li><li>■ <b>Axis:</b> number of the external axis (1 ... 6) (\$ETx_AX)</li><li>■ <b>Name of ext. kinematic system:</b> (\$ETx_NAME)</li></ul> <p>(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.)</p> <ol style="list-style-type: none"><li>3. Move the linear unit with the jog key “+”.</li><li>4. Specify whether the linear unit has moved to “+” or “-”. Confirm with <b>Next</b>.</li><li>5. Move the TCP to the reference point.</li><li>6. Press <b>Calibrate</b>.</li><li>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.</li><li>8. Press <b>Save</b>. The calibration data are saved.</li><li>9. The system asks whether the positions that have already been taught are to be corrected.</li></ol>

- If no positions have been taught prior to the calibration, it makes no difference whether the question is answered with **Yes** or **No**.
- If positions have been taught prior to the calibration:  
Answering **Yes** will cause positions with base 0 to be corrected automatically. Other positions will not be corrected.  
Answering **No** will cause no positions to be corrected.

**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.

### 5.12.6.3 Entering the linear unit numerically

#### Precondition

- 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)
- Operating mode T1

#### Procedure

1. In the main menu, select **Start-up > Calibrate > External kinematic system > 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.
6. The system asks whether the positions that have already been taught are to be corrected.
  - If no positions have been taught prior to the calibration, it makes no difference whether the question is answered with **Yes** or **No**.
  - If positions have been taught prior to the calibration:  
Answering **Yes** will cause positions with base 0 to be corrected automatically. Other positions will not be corrected.  
Answering **No** will cause no positions to be corrected.

**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.

### 5.12.7 Calibrating an external kinematic system

#### 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.  
 (>>> 5.12.6 "Linear unit" Page 141)

#### Overview

Calibration of an external kinematic system consists of 2 steps:

Step	Description
1	<p>Calibrate the root point of the external kinematic system.      (&gt;&gt;&gt; 5.12.7.1 "Calibrating the root point" Page 144)</p> <p>If the calibration data are already known, they can be entered directly.      (&gt;&gt;&gt; 5.12.7.2 "Entering the root point numerically" Page 146)</p>
2	<p>If there is a workpiece on the external kinematic system: calibrate the base of the workpiece.      (&gt;&gt;&gt; 5.12.7.3 "Calibrating a workpiece base" Page 146)</p> <p>If the calibration data are already known, they can be entered directly.      (&gt;&gt;&gt; 5.12.7.4 "Entering the workpiece base numerically" Page 148)</p> <p>If there is a tool mounted on the external kinematic system: calibrate the external tool.      (&gt;&gt;&gt; 5.12.7.5 "Calibrating an external tool" Page 148)</p> <p>If the calibration data are already known, they can be entered directly.      (&gt;&gt;&gt; 5.12.7.6 "Entering the external tool numerically" Page 149)</p>

#### 5.12.7.1 Calibrating the root point

#### Description

In order to be able to move the robot with a mathematical coupling to a kinematic system, the robot must know the precise location of the kinematic system. This location is determined by means of root point calibration.

The TCP of a tool that has already been calibrated is moved to a reference point on the kinematic system 4 times. The position of the reference point 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 point to calculate the root point of the kinematic system.

In the case of external kinematic systems from KUKA, the reference point is configured in the system variable \$ET<sub>x</sub>\_TPINFL in the machine data. This contains the position of the reference point relative to the FLANGE coordinate system of the kinematic system. ( $x$  = number of the kinematic system.) The reference point is also marked on the kinematic system. During calibration, this reference point must be addressed.

In the case of non-KUKA external kinematic systems, the reference point must be configured in the machine data.



**Fig. 5-32: Root point calibration principle**

#### Precondition

- The machine data of the kinematic system have been configured and loaded into the robot controller.
- A previously calibrated tool is mounted on the mounting flange.
- If \$ET<sub>x</sub>\_TPINFL is to be modified: user group “Expert”
- Operating mode T1

#### Procedure

1. In the main menu, select **Start-up > Calibrate > External kinematic system > Root point**.
2. Select the external kinematic system and assign a new kinematic name if required. Confirm with **Next**.
3. Select the number of the tool that has already been calibrated. Confirm with **Next**.
4. The value of \$ET<sub>x</sub>\_TPINFL is displayed.
  - If the value is not correct: the value can be modified here in the user group “Expert”.
  - If the value is correct: confirm with **Next**.
5. Move the TCP to the reference point.
6. Press **Calibrate**. Confirm with **Next**.
7. Repeat steps 5 and 6 three times. Each time, move the kinematic system first so that the reference point is approached from different positions.
8. Press **Save**.

### 5.12.7.2 Entering the root point numerically

**Precondition**

- The machine data of the kinematic system have been configured and loaded into the robot controller.
- The following numerical values are known, e.g. from CAD data:
  - Distance between the origin of the ROOT coordinate system and the origin of the WORLD coordinate system (X, Y, Z)
  - Orientation of the ROOT coordinate system relative to the WORLD coordinate system (A, B, C)
- Operating mode T1

**Procedure**

1. In the main menu, select **Start-up > Calibrate > External kinematic system > Root point (numeric)**.
2. Select the external kinematic system and enter a new kinematic name if required. Confirm with **Next**.  
(The name is automatically also assigned to the BASE coordinate system.)
3. Enter the data of the ROOT coordinate system. Confirm with **Next**.
4. Press **Save**.

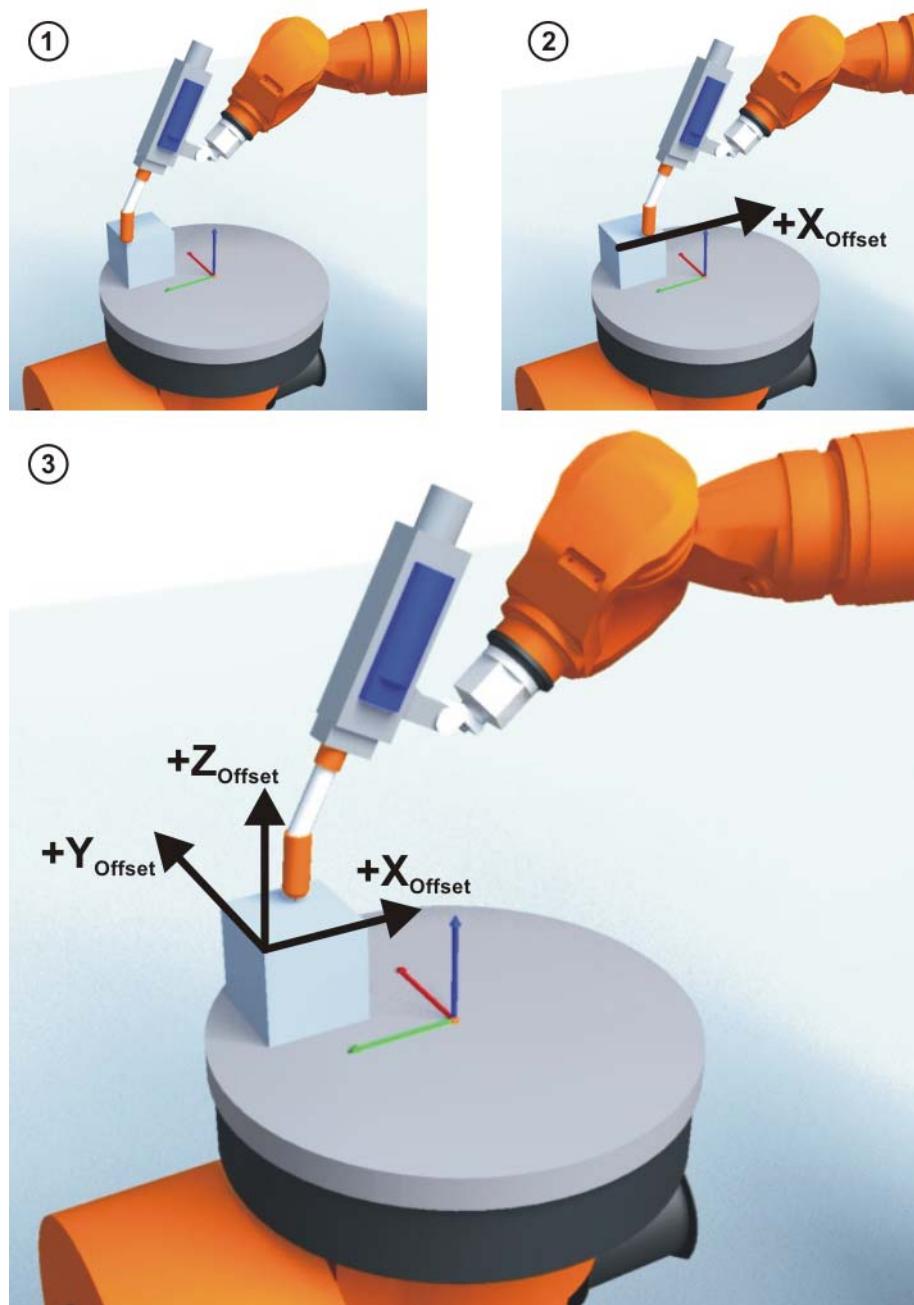
### 5.12.7.3 Calibrating a workpiece base

**Description**

During this calibration, the user assigns a BASE coordinate system to a workpiece located on the kinematic system. This BASE coordinate system is relative to the FLANGE coordinate system of the kinematic system. The base is thus a moving base that moves in the same way as the kinematic system.

It is not strictly necessary to calibrate a base. If none is calibrated, the FLANGE coordinate system of the kinematic system is taken as the base.

During calibration, the TCP of a calibrated tool is moved to the origin and 2 other points of the desired base. These 3 points define the base. Only one base can be calibrated per kinematic system.



**Fig. 5-33: Base calibration principle**

#### Precondition

- The machine data of the kinematic system have been configured and loaded into the robot controller.
- A previously calibrated tool is mounted on the mounting flange.
- The root point of the external kinematic system has been calibrated.
- Operating mode T1

#### Procedure

1. In the main menu, select **Start-up > Calibrate > External kinematic system > Offset**.
2. Select the BASE coordinate system for which the offset is to be calibrated. Confirm with **Next**.
3. Select the external kinematic system. Confirm with **Next**.
4. Select the number of the tool that has already been calibrated. Confirm with **Next**.
5. Move the TCP to the origin of the workpiece base. Press **Calibrate** and confirm with **Next**.

6. Move the TCP to a point on the positive X axis of the workpiece base. Press **Calibrate** and confirm with **Next**.
7. Move the TCP to a point in the XY plane with a positive Y value. Press **Calibrate** and confirm with **Next**.
8. Press **Save**.

#### 5.12.7.4 Entering the workpiece base numerically

##### Precondition

- The machine data of the kinematic system have been configured and loaded into the robot controller.
- The following numerical values are known, e.g. from CAD data:
  - Distance between the origin of the workpiece base and the origin of the FLANGE coordinate system of the kinematic system (X, Y, Z)
  - Rotation of the axes of the workpiece base relative to the FLANGE coordinate system of the kinematic system (A, B, C)
- The root point of the external kinematic system has been calibrated.
- Operating mode T1

##### Procedure

1. In the main menu, select **Start-up > Calibrate > External kinematic system > Offset (numeric)**.
2. Select the BASE coordinate system for which the offset is to be calibrated. Confirm with **Next**.
3. Select the external kinematic system. Confirm with **Next**.
4. Enter data. Confirm with **Next**.
5. Press **Save**.

#### 5.12.7.5 Calibrating an external tool

##### Description

During calibration of the external tool, the user assigns a coordinate system to the tool mounted on the kinematic system. This coordinate system has its origin in the TCP of the external tool and is relative to the FLANGE coordinate system of the kinematic system.

First of all, the user communicates to the robot controller the TCP of the tool mounted on the kinematic system. This is done by moving a calibrated tool to the TCP.

Then, the orientation of the coordinate system of the tool is communicated to the robot controller. For this purpose, the user aligns the coordinate system of the calibrated tool parallel to the new coordinate system. There are 2 variants:

- **5D**: The user communicates the tool direction to the robot controller. By default, the tool direction is the X axis. The orientation of the other axes is defined by the system and cannot be influenced by the user.  
The system always defines the orientation of the other axes in the same way. If the tool subsequently has to be calibrated again, e.g. after a crash, it is therefore sufficient to define the tool direction again. Rotation about the tool direction need not be taken into consideration.
- **6D**: The user communicates the direction of all 3 axes to the robot controller.



If **6D** is selected: it is advisable to document the alignment of all axes. If the tool subsequently has to be calibrated again, e.g. after a crash, the axes must be aligned the same way as the first time in order to be able to continue moving to existing points correctly.

The robot controller saves the coordinates of the external tool as the BASE coordinate system.

- Precondition**
- The machine data of the kinematic system have been configured and loaded into the robot controller.
  - A previously calibrated tool is mounted on the mounting flange.
  - The root point of the external kinematic system has been calibrated.
  - Operating mode T1



The following procedure applies if the tool direction is the default tool direction (= X axis). If the tool direction has been changed to Y or Z, the procedure must also be changed accordingly.

(>>> 5.12.1 "Defining the tool direction" Page 126)

- Procedure**
1. In the main menu, select **Start-up > Calibrate > Fixed tool > External kinematic offset**.
  2. Select the number of the BASE coordinate system in which the coordinates of the external tool are saved and assign a name for the BASE coordinate system. Confirm with **Next**.
  3. Select the external kinematic system. Confirm with **Next**.
  4. Select the number of the tool that has already been calibrated. Confirm with **Next**.
  5. Select a variant in the **5D / 6D** box. Confirm with **Next**.
  6. Move the TCP of the calibrated tool to the TCP of the external tool. Press **Calibrate** and confirm with **Next**.
  7. If **5D** is selected:  
Align  $+X_{BASE}$  parallel to  $-Z_{FLANGE}$ .  
(i.e. align the mounting flange perpendicular to the tool direction of the external tool.)  
If **6D** is selected:  
Align the mounting flange so that its axes are parallel to the axes of the external tool:
    - $+X_{BASE}$  parallel to  $-Z_{FLANGE}$   
(i.e. align the mounting flange perpendicular to the tool direction of the external tool.)
    - $+Y_{BASE}$  parallel to  $+Y_{FLANGE}$
    - $+Z_{BASE}$  parallel to  $+X_{FLANGE}$
  8. Press **Calibrate** and confirm with **Next**.
  9. Press **Save**.

#### 5.12.7.6 Entering the external tool numerically

- Precondition**
- The following numerical values are known, e.g. from CAD data:
    - Distance between the TCP of the external tool and the origin of the FLANGE coordinate system of the kinematic system (X, Y, Z)
    - Rotation of the axes of the external tool relative to the FLANGE coordinate system of the kinematic system (A, B, C)
  - Operating mode T1
- Procedure**
1. In the main menu, select **Start-up > Calibrate > Fixed tool > Numeric input**.
  2. Select a number for the external tool and assign a tool name. Confirm with **Next**.
  3. Enter data. Confirm with **Next**.
  4. Press **Save**.

## 5.13 Load data

The load data are factored into the calculation of the paths and accelerations and help to optimize the cycle times. The load data must be entered in the robot controller.

### Sources

Load data can be obtained from the following sources:

- Software option KUKA.LoadDataDetermination (only for payloads on the flange)
- Manufacturer information
- Manual calculation
- CAD programs

### 5.13.1 Checking loads with KUKA.Load

All load data (payload and supplementary loads) must be checked with the KUKA.Load software. Exception: If the payload is checked with KUKA.LoadDataDetermination, it is not necessary to check it with KUKA.Load.

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](http://www.kuka.com).



More information is contained in the **KUKA.Load** documentation.

### 5.13.2 Calculating payloads with KUKA.LoadDataDetermination

#### Description

KUKA.LoadDataDetermination can be used to calculate payloads exactly and transfer them to the robot controller.

#### Precondition

- T1 or T2 operating mode
- No program is selected.

#### Procedure

- In the main menu, select **Start-up > Service > Load data determination**.



More information is contained in the **KUKA.LoadDataDetermination** documentation.

### 5.13.3 Entering payload data

#### Description

For a tool or workpiece on the flange, the payload data must be communicated to the robot controller.

The payload data can be entered numerically or determined with KUKA.LoadDataDetermination and transferred to the robot controller.

#### Precondition

- The payload data have been checked with KUKA.Load or KUKA.LoadDataDetermination and the robot is suitable for these payloads.

#### Procedure

1. In the main menu, select **Start-up > Calibrate > Tool > Tool load data**.
2. In the **Tool no.** box, select the number of the tool. Confirm with **Next**.
3. Enter the payload data:
  - Box **M**: Mass
  - Boxes **X**, **Y**, **Z**: Position of the center of gravity relative to the flange

- Boxes **A**, **B**, **C**: Orientation of the principal inertia axes relative to the flange
  - Boxes **JX**, **JY**, **JZ**: Mass moments of inertia  
(**JX** is the inertia about the X axis of the coordinate system that is rotated relative to the flange by A, B and C. **JY** and **JZ** are the analogous moments of inertia about the Y and Z axes.)  
Or, if the default values for this robot type are to be used: press **Default**.
4. If online load data checking is available (this depends on the robot type): configure as required.  
(>>> 5.13.5 "Online load data check (OLDC)" Page 151)
  5. Confirm with **Next**.
  6. Press **Save**.

#### 5.13.4 Entering supplementary load data

**Description** The supplementary load data must be entered in the robot controller.

Reference systems of the X, Y and Z values for each supplementary load:

Load	Reference system
Supplementary load A1	ROBROOT coordinate system $A1 = 0^\circ$
Supplementary load A2	ROBROOT coordinate system $A2 = -90^\circ$
Supplementary load A3	FLANGE coordinate system $A4 = 0^\circ, A5 = 0^\circ, A6 = 0^\circ$

**Precondition** ■ The supplementary loads have been verified with KUKA.Load and are suitable for this robot type.

- Procedure**
1. In the main menu, select **Start-up > Calibrate > Supplementary load data**.
  2. Enter the number of the axis on which the supplementary load is to be mounted. Confirm with **Continue**.
  3. Enter the load data. Confirm with **Continue**.
  4. Press **Save**.

#### 5.13.5 Online load data check (OLDC)

**Description** For many robot types, the robot controller monitors whether or not there is an overload or underload during operation. This monitoring is called "Online load data check" (OLDC).

If the OLDC 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 153)

OLDC is available for those robot types for which KUKA.LoadDataDetermination can also be used. Whether or not OLDC is available for the current robot type can be checked by means of **\$LDC\_LOADED** (TRUE = yes).

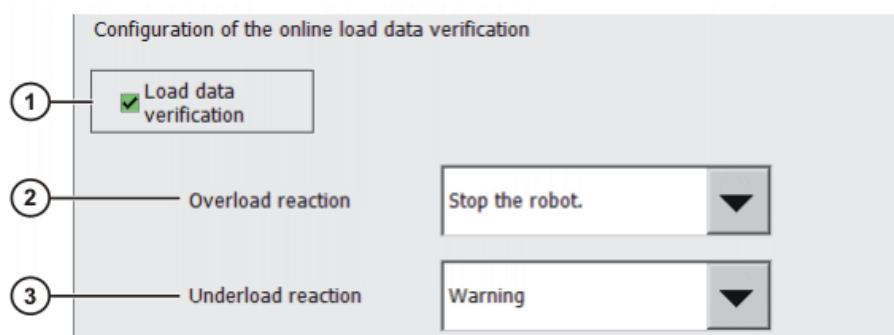
Overload	There is an overload if the actual load is greater than the configured load.
Underload	There is an underload if the actual load is less than the configured load.

**Configuration**

OLDC can be configured as follows:

- During manual entry of the tool data  
(>>> 5.12.2.5 "Numeric input" Page 132)
- During separate entry of the payload data  
(>>> 5.13.3 "Entering payload data" Page 150)

The following boxes are displayed in the same window in which the payload data are also entered:



**Fig. 5-34: Online load data check**

Item	Description
1	<p><b>TRUE:</b> OLDC is activated for the tool displayed in the same window. The defined reactions are carried out in the case of an overload or underload.</p> <p><b>FALSE:</b> OLDC is deactivated for the tool displayed in the same window. There is no reaction in the case of an overload or underload.</p>
2	<p>The overload reaction can be defined here.</p> <ul style="list-style-type: none"> <li>■ <b>None:</b> No reaction.</li> <li>■ <b>Warning:</b> The robot controller generates the following status message: <i>Check of robot load (Tool {No.}) calculated overload</i>.</li> <li>■ <b>Stop robot:</b> The robot controller generates an acknowledgement message with the same content as that generated under <b>Warning</b>. The robot stops with a STOP 2.</li> </ul>
3	<p>The underload reaction can be defined here. The possible reactions are analogous to those for an overload.</p>

The reactions can be modified in the KRL program using the system variable \$LDC\_CONFIG. (>>> "\$LDC\_CONFIG" Page 153)

**NULFRAME**

OLDC cannot be configured for motions to which the tool **NULFRAME** has been assigned. 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
<i>Index</i>	Type: INT Tool number ■ 1 ... 32
<i>Reaction</i>	Type: CHAR ■ #NONE (= <b>None</b> ) ■ #WARNONLY (= <b>Warning</b> ) ■ #STOPROBOT (= <b>Stop robot</b> )

**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 <b>None</b> .
4	The reaction is set to <b>Warning</b> . If there is an underload or overload, the corresponding status messages will be displayed in the message window.

**\$LDC\_RESULT**       $\$LDC\_RESULT[Index] = Result$

Element	Description
<i>Index</i>	Type: INT Tool number ■ 1 ... 32
<i>Result</i>	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.

## 5.14 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.

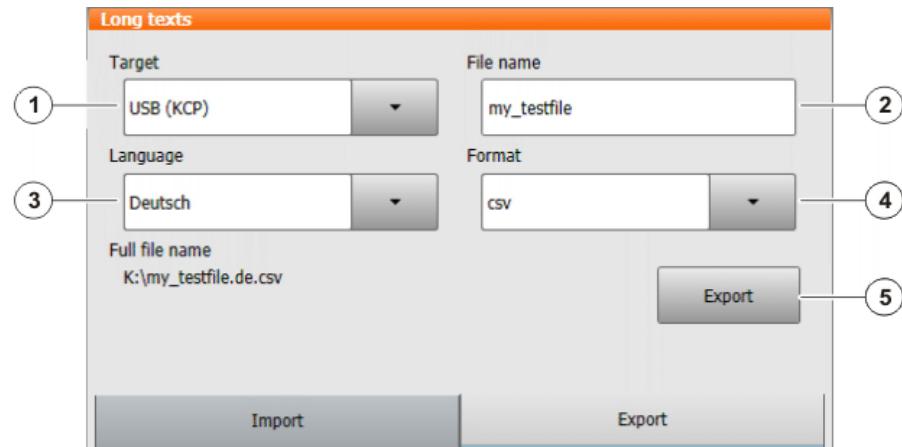
A file that originated as a long text export is automatically structured in such a way that it can be re-imported. If a file is to be filled with names manually, it is advisable first to assign a few dummy long texts in the robot controller, then to perform an export and fill the exported file.

**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. 5-35: Exporting long texts**

Item	Description
1	Select the destination for the exported file. The entry <b>Network</b> is only available here if a path has been configured in the <b>Robot data</b> window.
2	Specify the desired file name. If <b>Network</b> is selected under item 1, the archive name configured in the <b>Robot data</b> window is displayed. The name can be changed here. This does not change it in the <b>Robot data</b> 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 <b>All languages</b> .
4	Select the desired file format.
5	Starts the export.

## "Import" tab

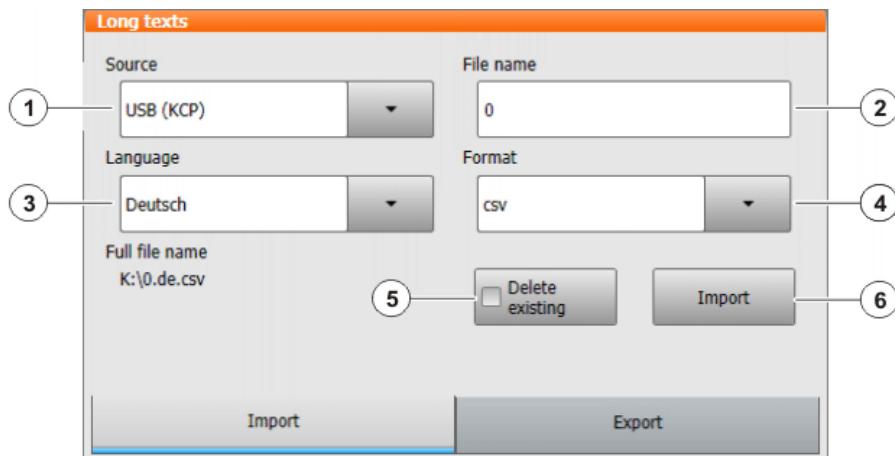


Fig. 5-36: Importing long texts

Item	Description
1	Specify the source from which files are to be imported. The entry <b>Network</b> is only available here if a path has been configured in the <b>Robot data</b> window.
2	Specify the name of the file to be imported, but without the language suffix. If <b>Network</b> is selected under item 1, the archive name configured in the <b>Robot data</b> window is displayed. The name can be changed here. This does not change it in the <b>Robot data</b> window.
3	Specify the language matching the language suffix of the file.
4	Specify the format of the file.
5	<ul style="list-style-type: none"> <li>■ <b>Activated:</b> All existing long texts are deleted. The contents of the file are applied.</li> <li>■ <b>Deactivated:</b> Entries in the file overwrite existing long texts. Existing long texts for which there is no entry in the file are retained.</li> </ul>
6	Starts the import.

## 5.15 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.

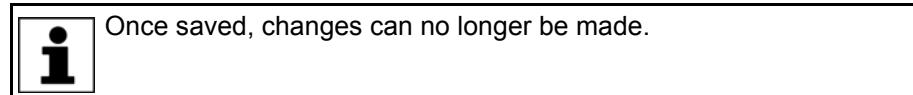


The controller variant "KR C4 compact" has no Controller System Panel and no flashing lights to indicate when maintenance work is due.

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.

### 5.15.1 Logging maintenance

**Description** It is not possible to log multiple maintenance activities of the same kind on one day.



**Precondition**

- “Expert” user group

**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. 5-37: Maintenance input**

Item	Description
1	Select which type of maintenance has been carried out.
2	Enter who performed the maintenance.

Item	Description
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 types**

By default, the following maintenance types can be selected:

- **Basic inspection**
- **In-line wrist maintenance**
- **Main axis maintenance**
- **Gear backlash measurement**
- **Minor electrical maintenance**
- **Major electrical maintenance**
- **Data backup with spare hard drive**
- **Repair**

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.

### 5.15.2 Displaying a maintenance log

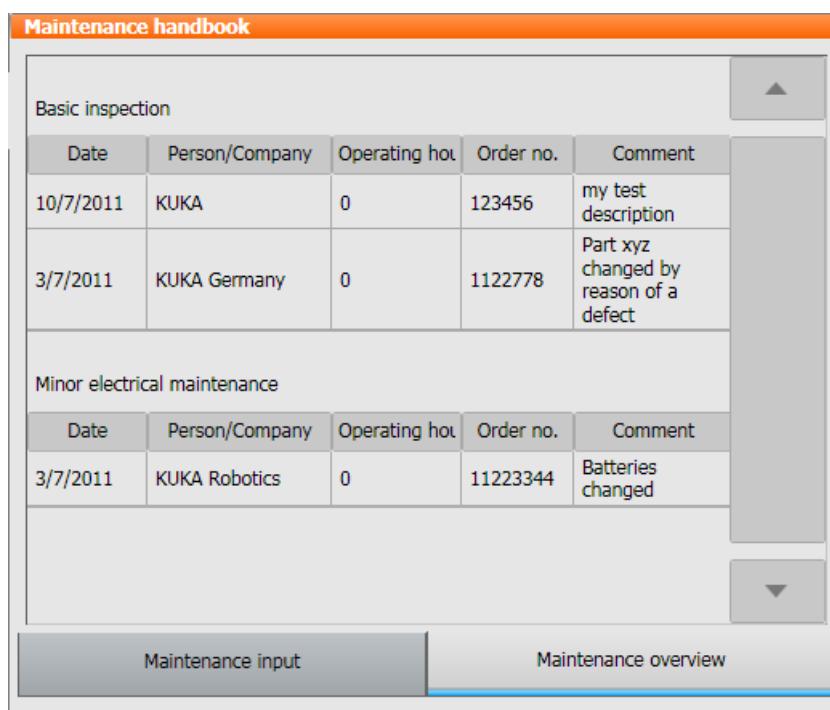
**Description**

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. 5-38: Maintenance overview**



## 6 Configuration

### 6.1 Configuring the KUKA Line Interface (KLI)

**Description** The KLI is the Ethernet interface of the robot controller for external communication. It is a physical interface and can contain multiple virtual interfaces.

In order for external PCs to be able to connect to the robot controller via a network, the KLI must be configured accordingly. This is a precondition, for example, for being able to transfer WorkVisual projects to the robot controller via the KLI.



The following address ranges are used by default by the robot controller for internal purposes. IP addresses from this range cannot therefore be assigned by the user via the smartHMI. The system indicates an error.

- 169.254.0.0 ... 169.254.255.255
- 172.16.0.0 ... 172.16.255.255
- 172.17.0.0 ... 172.17.255.255
- 192.168.0.0 ... 192.168.0.255

#### Field buses

How the KLI has to be configured depends, among other things, on whether an Ethernet-based field bus is installed on the robot controller.

Ethernet-based field buses are:

- KR C4 PROFINET
- KR C4 EtherNet/IP

#### 6.1.1 Configuring the Windows interface (without field bus)

**Description** The Windows interface is a virtual interface of the KLI. It has a preconfigured static IP address:

- IP address: 172.31.1.147
- Subnet mask: 255.255.0.0

These values can be modified by the user. If required, it is also possible to switch to dynamic address assignment. Similarly, it is also possible to switch back from a dynamic address to a static address.

#### DNS Server:

If required, a DNS server address can be specified.

If the robot controller is connected to the IT network, name resolution of the devices can be carried out via the DNS server address. (In other words, the DNS server receives a query with a device name and responds with the corresponding IP address.)

“0.0.0.0” is admissible and is ignored by the system.

#### Precondition

- There is no Ethernet-based field bus installed on the robot controller.
- If the address type **Dynamic IP address** is to be selected: there is a DHCP server present in the network.
- No program is selected.
- T1 or T2 mode
- User group “Expert”

**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 163)

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.

**Network configuration**

Windows interface (virtual5)

Address type:	Fixed IP address
IP address:	172 . 31 . 1 . 147
Subnet mask:	255 . 255 . 0 . 0
Standard gateway:	0 . 0 . 0 . 0
DNS Server:	0 . 0 . 0 . 0
Advanced...	

Fig. 6-1: Example: Fixed IP address

## 6.1.2 Configuring the field bus interface and creating the Windows interface

**Description** If an Ethernet-based field bus is installed on the robot controller, it is automatically assigned virtual interface “virtual5”. This must be adapted to the field bus.

“Virtual5” can simultaneously be used as a Windows interface, but the static IP address of the field bus must then be used for Windows access.

If a different IP configuration is to be used for the Windows interface, an additional virtual interface, e.g. “virtual6”, must be added and configured for this purpose.

- Precondition**
- No program is selected.
  - T1 or T2 mode
  - User group “Expert”
  - If KR C4 PROFINET is installed: the PROFINET device has been named.



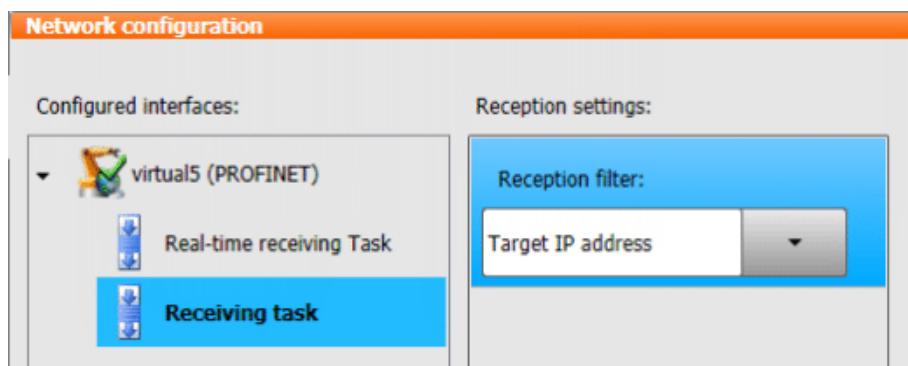
Information about device naming can be found in the documentation  
**KR C4 PROFINET**.

**Procedure**



If an address or subnet field is framed in red, this indicates the presence of an error.  
(>>> 6.1.5 "Error display in the Address and Subnet boxes"  
Page 163)

1. In the main menu, select **Start-up > Network configuration**. The **Network configuration** window opens. The “virtual5” interface is displayed.
2. If it has not already been selected: select the type **Fixed IP address** in the **Address type:** box.
3. Complete the boxes **IP address:** and **Subnet mask:**. Enter the address and mask that are also assigned by the PLC of the field bus.
4. Press **Advanced....** The window for advanced network configuration opens.
5. Select the **Interfaces** tab.
6. Select the entry “virtual5” in the **Configured interfaces:** area and enter the field bus (e.g. “PROFINET”) in the **Interface designation:** box.
7. Several “Reception task” entries are displayed under the entry “virtual5”. Select the bottom one.
8. The **Reception filter:** box indicates **Accept all**. Change the entry to **Target IP address**.



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

9. Select the entry “virtual5” and press the **Add interface** button. The entry “virtual6” is automatically created.

10. Select the entry “virtual6” and enter “WINDOWS” in the **Interface designation:** box.
11. Select the **Dynamic IP address** type in the **Address type:** box.

**Fixed IP address** can also be selected here if a static IP address is desired. The static address must be in a different address range, however, than the address of “virtual5”.

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

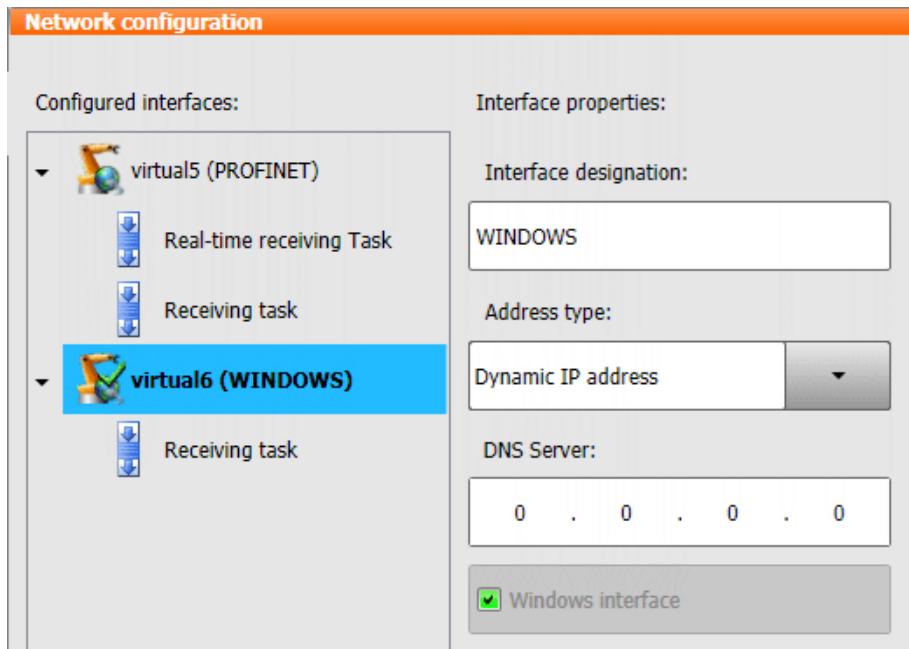


Fig. 6-3: “virtual6” interface, with “PROFINET” as example

13. Select the “Reception task” entry under the “virtual6” entry.
14. If not already the case, change the **Reception filter:** box to **Accept all**.
15. Press **Save**.
16. Close the **Network configuration** window using the Close icon.
17. Reboot the robot controller. For this, select **Shutdown** in the main menu and select the option **Reload files**.

The check mark in the **Windows interface** check box cannot be removed. De-selection is only possible by defining a different interface as the Windows interface.

### 6.1.3 Displaying ports of the Windows interface or enabling an additional port



It is not generally necessary to enable additional ports. If this is nevertheless to be done, KUKA Deutschland GmbH must be contacted beforehand.

#### NOTICE

The ports enabled by KUKA by default must not be removed. Doing so may result in the loss of functions of the robot controller.

#### Precondition

- User group “Expert”
- Operating mode T1 or T2.
- No program is selected.

- |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Procedure</b> | <ol style="list-style-type: none"> <li>1. In the main menu, select <b>Start-up &gt; Network configuration</b>. The <b>Network configuration</b> window opens.</li> <li>2. Press <b>Advanced....</b> The window for advanced network configuration opens.</li> <li>3. Select the <b>NAT</b> tab. A list of all the enabled ports of the Windows interface is displayed in the <b>Available ports:</b> area.</li> <li>4. If a port is to be enabled:           <ol style="list-style-type: none"> <li>a. Press <b>Add port</b>. A new port with the number "0" is added to the list.</li> <li>b. Complete the boxes <b>Port number:</b> and <b>Permitted protocols:</b>.</li> <li>c. Press <b>Save</b>.</li> </ol> <p>A maximum total of 40 ports can be enabled.</p> </li> <li>5. Close the <b>Network configuration</b> window using the Close icon.</li> <li>6. If modifications have been made: Reboot the robot controller with the setting <b>Reload files</b>.</li> </ol> |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

#### 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.

- |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Precondition</b> | <ul style="list-style-type: none"> <li>■ User group "Expert"</li> <li>■ Operating mode T1 or T2.</li> <li>■ No program is selected.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Procedure</b>    | <ol style="list-style-type: none"> <li>1. In the main menu, select <b>Start-up &gt; Network configuration</b>. The <b>Network configuration</b> window opens.</li> <li>2. Press <b>Advanced....</b> The window for advanced network configuration opens.</li> <li>3. Select the <b>User-defined filters</b> tab. The filters of the KUKA Line Interface and their properties are displayed here.</li> <li>4. If modifications are required: Carry these out and press <b>Save</b>.</li> <li>5. Close the <b>Network configuration</b> window using the Close icon.</li> <li>6. If modifications have been made: Reboot the robot controller with the setting <b>Reload files</b>.</li> </ol> |

#### 6.1.5 Error display in the Address and Subnet boxes

- |                    |                                                                                                                            |
|--------------------|----------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | 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 <b>Internal subnets</b> tab are displayed with a red frame.	<ul style="list-style-type: none"> <li>■ Change the actual address.</li> <li>■ Or, only after consultation with KUKA: change the address of the internal subnet.</li> </ul> The red frame disappears.

**NOTICE**

The subnet configuration of the robot controller may be modified only in consultation with KUKA Deutschland GmbH. Modifications carried out without consultation may result in the loss of functions of the robot controller.

**Example****Example of an entry that is not system-compliant:**

In the binary display, subnet masks may only contain closed groups of leading ones.

- The subnet mask “255.255.208.0” is entered.
- The box is now displayed with a red frame.  
Reason: the binary representation of “208” corresponds to “11010000” and is thus not a valid entry.
- Possible remedy: Enter 255.255.240.0.  
The binary representation of “240” corresponds to “11110000” and is thus a valid entry.

## 6.2 Displaying the subnet configuration of the robot controller

**Description**

The subnet configuration of the robot controller can be displayed. This makes it possible to compare it with the subnets of the customer network.

**NOTICE**

The subnet configuration of the robot controller may be modified only in consultation with KUKA Deutschland GmbH. Modifications carried out without consultation may result in the loss of functions of the robot controller.

**Precondition**

- User group “Expert”
- Operating mode T1 or T2.
- No program is selected.

**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 163)



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 group “Expert”
- Operating mode T1 or T2.
- 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

**Precondition**

- User group “Safety recovery”
- Operating mode T1 or T2

**Procedure**

In order to be able to save the configuration values, a reconfiguration must be carried out. During reconfiguration, all outputs are briefly set to zero before returning to their original state.

1. In the main menu, select **Configuration > Safety configuration**.  
The **Safety configuration** window opens.
2. Select the **Axis monitoring** tab.
3. Edit the parameters as required and press **Save**.
4. Answer the request for confirmation with **Yes**. The controller is reconfigured.
5. Once the reconfiguration has been completed, the following message is displayed: *The changes were saved successfully*.

Confirm the message with **OK**.



**WARNING** Following modifications to the safety configuration, the values for the safe axis monitoring functions must be checked.  
 (>>> 6.5 "Checking the values for the safe axis monitoring functions"  
 Page 170)



**WARNING** 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.  
 (>>> 6.4.3 "Checking the limits for the maximum axis velocity in T1 mode"  
 Page 169)

#### Editable parameters

The following parameters can be set for each axis. It is not generally necessary to change the default values, however.

Parameter	Description
<b>Braking time</b>	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 166)
<b>Maximum velocity T1</b>	Maximum velocity in T1 <ul style="list-style-type: none"> <li>■ Rotational axes: <b>1.00°... 100.00°/s</b>            Default: 30 °/s</li> <li>■ Linear axes: <b>1.00 ... 1,500.00 mm/s</b>            Default: 250 mm/s</li> </ul> <p>This parameter enables a servo gun, for example, to be calibrated in T1 with a higher velocity than 250 mm/s.</p> <p><b>Note:</b> The Cartesian velocities at the flange and at the TCP are monitored independently of this parameter and cannot exceed 250 mm/s.</p> <p>(&gt;&gt;&gt; 6.4.2 "Parameter Maximum velocity T1 for couplable axes" Page 168)</p>
<b>Position tolerance</b>	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. <ul style="list-style-type: none"> <li>■ Rotational axes: <b>0.001 ... 1 °</b>            Default: 0.01 °</li> <li>■ Linear axes: <b>0.003 - 3 mm</b>            Default: 0.1 mm</li> </ul>

#### 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 monitoring ramp can be specified using the parameter **Braking time**.



The parameter **Braking time** modifies the monitoring ramp. It does not modify the actual time required by the kinematic system for braking.



**WARNING** Only alter 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 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.

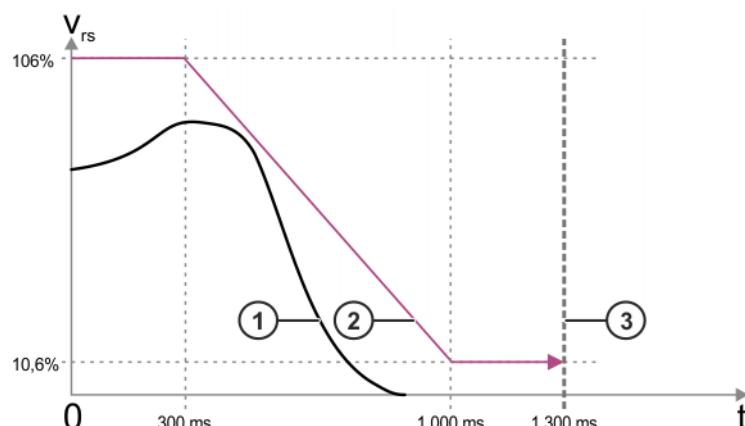
The monitoring ramp is determined as follows:

- The robot controller subtracts 200 ms from the value of the parameter **Braking time** (taking into account the brake closing time). The result is the monitoring time. For example, the default value of 1 500 ms results in a monitoring time of 1 300 ms.

When this time has elapsed, another monitoring function begins:

- The ramp has plateaus of 300 ms at the start and end.

The plateau at the start is always 106% of the rated speed of the axis. The plateau at the end is always 10.6 %.



**Fig. 6-5: Monitoring ramp**

- 1 Velocity profile during braking (example)
- 2 Monitoring ramp (default value **Braking time** 1 500 ms)
- 3 From this moment on, standstill monitoring begins.

**V<sub>rs</sub>** Rated speed of the axis (rs = "rated speed")

**t** Time

The value "0" on the time axis is the moment at which the safety stop 1 or 2 begins.

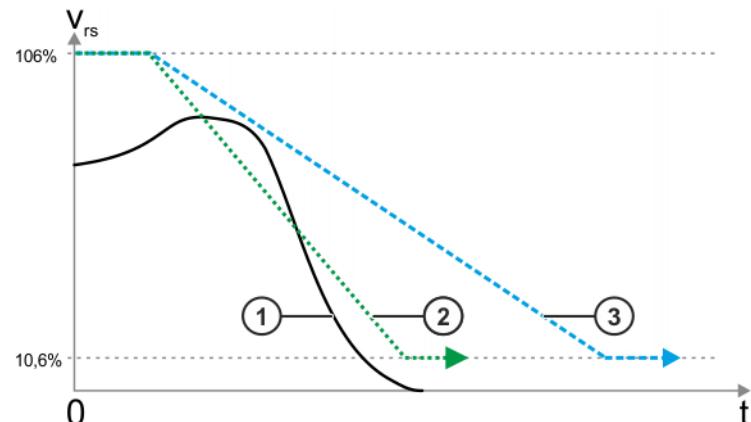
## Limitations

- **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.
- The parameter **Braking time** usually has no effect in T1, since it refers to the axis-specific monitoring. In T1, however, there is another (non-configurable) monitoring function for the Cartesian velocity on the flange. This is usually stricter.

**Value increased**

If the value **Braking time** is increased, this has the following effect:

The monitoring ramp becomes longer and flatter, i.e. monitoring is now less strict. There is now a lower probability that a 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 Parameter Maximum velocity T1 for couplable axes

**Description**

The safety controller monitors whether the maximum velocity in T1 remains below the configured values even for axes which are configured as couplable or grouped together in coupling groups. Axes which are configured as couplable or grouped together in coupling groups are not displayed in the safety configuration. To be able to alter the configured values for these axes, the coupling must be temporarily canceled.



**WARNING** Only alter the default value if it is necessary to do so. This can be the case, for example, when positioning welding guns if these are to be moved at process velocity in T1 mode. The safety recovery technician must check whether and to what extent the **Maximum velocity T1** 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.



**WARNING** 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.  
(>>> 6.4.3 "Checking the limits for the maximum axis velocity in T1 mode" Page 169)



The coupling of the axes can be canceled in WorkVisual. Information about this can be found in the WorkVisual documentation. The parameter **Maximum velocity T1** can also be modified in WorkVisual.

### 6.4.3 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 INSTRUCTIONS** 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 169)
  - 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 15xxx 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 INSTRUCTIONS** 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 15xxx 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:

$$\$VEL\_AXIS[x] = (V_{\text{Test}} / V_{\text{max}}) * 100 = (25^{\circ}/\text{s} / 360^{\circ}/\text{s}) * 100 = 7$$

Element	Description
x	Number of the axis

Element	Description
V <sub>test</sub>	Desired test velocity (in this example, 25°/s) Unit: °/s
V <sub>max</sub>	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.



The values must always be checked if the checksum has changed on the **Common** tab in the **Safety configuration** window, i.e. not only if the values themselves have been changed, but if any changes have been made that affect the safety configuration. If this check is not carried out, the safety configuration may contain incorrect data. Death to persons, severe injuries or considerable damage to property may result.

### Precondition

- The values most recently saved for the parameters **Braking time** and **Position tolerance** are known.  
The most recently saved values can generally be found in a checklist, sign-off sheet, or similar.

### Procedure

#### SAFETY INSTRUCTIONS

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 (HnfHIp)** area in the **Module** box.  
Data are now displayed for this area.
3. Compare the values displayed for **Braking time** and **Position tolerance** with the most recently saved values.
4. Result:
  - If the values match: OK.  
Close the **Diagnostic monitor** window. No further action is necessary.
  - If the values do not match:

Enter and save the values again. If necessary, transfer the WorkVisual project to the robot controller again.

Then carry out the check again. KUKA must be contacted if the values still do not match.

## 6.6 Checking the safety configuration of the robot controller

**Description** The safety configuration of the robot controller must be checked in the following cases:

- After activation of a WorkVisual project on the robot controller
- Generally after changes to the machine data (independent of WorkVisual).

**WARNING** If the safety configuration is not checked and updated where necessary, it may contain incorrect data. Death, injuries or damage to property may result.

**Precondition**

- User group "Safety recovery"

**Procedure**

**SAFETY INSTRUCTIONS** 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 171)
  - c. The safety configuration also checks whether there are any other deviations (other than in the machine data) between the robot controller and the safety controller.  
If so, the **Troubleshooting wizard** window is opened. A description of the problem and a list of possible causes is displayed. The user can select the applicable cause. The wizard then suggests a solution.

**Deviations** The dialog message indicates which machine data in the robot controller deviate from those in the safety controller.

The message asks whether the safety configuration is to be updated, i.e. whether the machine data of the robot controller are to be applied to the safety configuration.

- If so: Answer the query with **Yes**.

**WARNING** If deviations are applied, the safety measures for start-up and recommissioning must then be carried out.

- If not: Answer the query with **No**.

**WARNING** In this case, the robot must not be operated. The machine data must be checked and corrected so that either the safety configuration detects no further deviations or the detected deviations can be applied.

## 6.7 Checksum of the safety configuration

<b>Description</b>	The checksum is updated each time the safety configuration is saved. The robot controller displays the checksum in the following locations: <ul style="list-style-type: none"><li>■ In the <b>Safety configuration</b> window on the <b>Common</b> tab : The window can be opened from the main menu by selecting <b>Configuration &gt; Safety configuration</b>.</li><li>■ For user group Expert or higher: In the file SCTLCRC.XML in the directory C:\KRC\ROBOTER\Config\User\Common</li></ul> The SCTLCRC.XML file is available so that, if required, the system integrator can read the checksum there via the higher-level controller.
--------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------



During start-up of the industrial robot, it is advisable to check that the value is correctly read from the SCTLCRC.XML file and transferred to the higher-level controller.

### SCTLCRC.XML

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <SafetyInfo ParameterCrc="3702332E" />
```

Fig. 6-7: Example: Checksum in the SCTLCRC.XML file

## 6.8 Exporting the safety configuration (XML export)

<b>Description</b>	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. <ul style="list-style-type: none"><li>■ 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.</li><li>■ If no safety option is installed on the robot controller, the parameters in the XML file are filled with default values (often "0").</li></ul>
--------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------



In addition to exporting, it is also possible to import a safety configuration when a safety option is installed. More detailed information about exporting and importing can be found in the safety option documentation.

It is also possible to import or export safety configurations in WorkVisual. Information about this can be found in the WorkVisual documentation.

### Procedure

1. In the main menu, select **Configuration > Safety configuration**.  
The **Safety configuration** window opens.
2. Press **Export**. The available drives are displayed.
3. Select the desired file path and press **Export**.  
The safety configuration is saved in an XML file. The file name is generated automatically.

## 6.9 Checking via PLC whether the safety configuration has been changed

<b>Description</b>	The activation code of the safety configuration can be stored on the PLC. Whenever the controller establishes a PROFIsafe connection to the PLC, the PLC returns the code to the safety controller. The safety controller compares
--------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

the received code with its current code. If the codes do not match, certain responses occur.

The comparison checks whether the local safety configuration has been modified.

If the codes do not match, the following responses occur on the robot controller.

- Safety stop 1 (if Start-up mode is not active)
- Message: *Error activation code comparison. PLC: {Stored activation code}, safety configuration: {Activation code of the safety configuration}*
- The safety controller signals the following PROFINET alarm to the PLC: *0x004B (75): Inconsistent iParameters (iParCRC error)*
- Message: *Safe device communication error {Device}*
- No PROFIsafe connection between controller and PLC

If the activation code on the PLC is changed, this results in a new PROFIsafe connection being established between the controller and the PLC. The PLC then sends the activation code to the safety controller again. If the correct code has now been sent, the robot can immediately be moved again.

#### **Precondition**

- Field bus: PROFINET
- Safety Interface: PROFIsafe
- A current device description file of the KUKA controller is present on the PLC.

#### **Preparation**

The activation code of the safety controller must be determined. There are two options for this.

Read activation code in diagnostic monitor:

1. Main menu **Diagnosis > Diagnostic monitor**
2. Select the **Safety controller (HnfHip)** area in the **Module** box.
3. The line **Activation code, controller** now displays the code.

or: Read activation code in safety controller:

The code is only displayed in the safety controller if a safety option has been installed, e.g. SafeOperation.

1. Main menu **Configuration > Safety configuration**
2. Tab **Common**, group **Parameter data set**, box **Activation code**

#### **Procedure**

Store an activation code in the PLC:

- On the PLC, enter the activation code manually in the PROFIsafe F parameter “F\_iPar\_CRC”.
- or: Transfer the activation code via a non-safe channel, e.g. via “I&M4 data”.



If the actual activation code of the safety configuration is 0x00, 0x01 must be entered on the PLC. 0x01 is the valid code for configurations with activation code 0x00 or 0x01.

#### **Start-up mode**

To deactivate the comparison for safety start-up, 0x00 can be entered as the activation code on the PLC. When the safety controller receives 0x00, no comparison is carried out.

#### **Diagnostic monitor**

In addition to the current activation code, the diagnostic monitor also displays the following information:

- The activation code received from the PLC
- The comparison status

Status	Meaning
Waiting for activation code	The safety controller has not yet received an activation code from the PLC.
Active, comparison successful	The safety controller has received an activation code from the PLC. It matches the code on the controller.
Active, comparison failed	The safety controller has received an activation code from the PLC. It does not match the code on the controller.
Deactivated	The safety controller has received the activation code 0x00 from the PLC. It does not perform a comparison.

**F parameters****Relevant F parameters:**

It is possible that not all parameters are visible on the configuration interface of the PLC.

Parameter	Description
F_iPar_CRC	Contains the activation code to be transferred. Default: 0, i.e. no comparison is carried out. (Corresponds to status = <b>deactivated</b> )
F_Block_ID	Must have the value 1 in order to enable performance of the comparison. If the value is 0, no evaluation of the iParameters is carried out, irrespective of the value in F_iPar_CRC. Default: 1
F_Check_iPar	Unchangeable setting: NoCheck “NoCheck” means that no further iParameters are transferred for checking. The value is not visible on most configuration interfaces.

## 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 10 groups is possible. A maximum of 25 variables per group is possible. System variables and user-defined variables can be displayed.

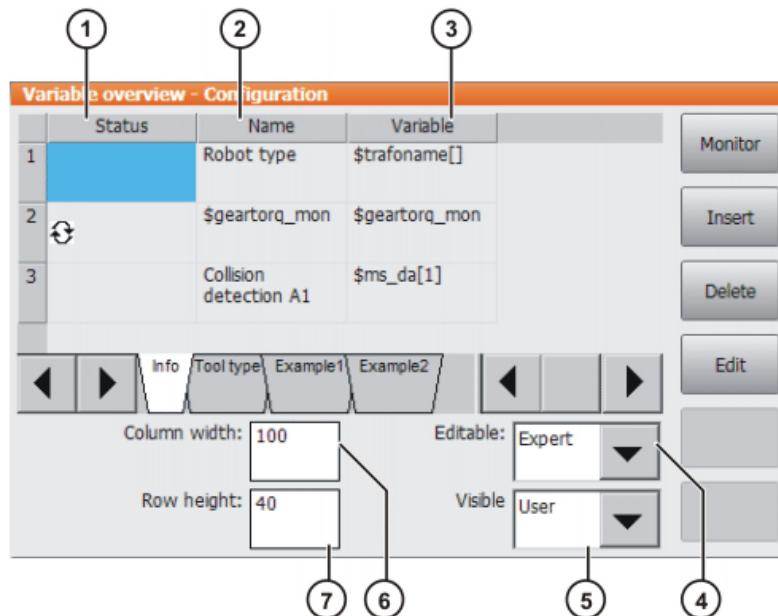
**Precondition**

- “Expert” user group

**Procedure**

1. In the main menu, select **Display > Variable > Overview > Configuration**.  
The **Variable overview – Configuration** window is opened.
2. Make the desired settings. To edit a cell, select it.
3. Press **OK** to save the configuration and close the window.

## Description



**Fig. 6-8: Variable overview - Configuration**

Item	Description
1	<ul style="list-style-type: none"> <li>■ Arrow symbol : If the value of the variable changes, the display is automatically refreshed.</li> <li>■ No arrow symbol: The display is not automatically refreshed.</li> </ul>
2	Descriptive name
3	Path and name of the variable <b>Note:</b> For system variables, the name is sufficient. Other variables must be specified as follows: <i>/R1/Program name/Variable name</i> 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. ( <b>&gt;&gt;&gt; 4.18.8 "Displaying the variable overview and modifying variables" Page 84</b> )

Button	Description
<b>Paste</b>	Displays additional buttons: <ul style="list-style-type: none"><li>■ <b>Row above:</b> Inserts a new row above the one currently selected.</li><li>■ <b>Row below:</b> Inserts a new row below the one currently selected.</li><li>■ <b>Group before:</b> Inserts a new group to the left of the one currently selected.</li><li>■ <b>Group after:</b> Inserts a new group to the right of the one currently selected.</li></ul>
<b>Delete</b>	Displays additional buttons: <ul style="list-style-type: none"><li>■ <b>Line:</b> The selected row is deleted.</li><li>■ <b>Group:</b> The current group is deleted.</li></ul>

## 6.11 Changing the password

### Procedure

1. Select **Configuration > User group** in the main menu. The current user group is displayed.
2. Press **Login....**
3. Select the user group for which the password is to be changed.
4. Press **Password ....**
5. Enter the old password. Enter the new password twice.  
For security reasons, the entries are displayed encrypted. Upper and lower case are taken into consideration.
6. Press **OK**. The new password is valid immediately.

## 6.12 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
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
<i>Level</i>	<p>Type: ENUM</p> <ul style="list-style-type: none"> <li>■ #OFF: Energy saving mode is deactivated.</li> <li>■ #LOW: Low saving</li> <li>■ #MIDDLE: Medium saving</li> <li>■ #HIGH: High saving</li> </ul>

## 6.13 Configuring workspaces

Workspaces can be configured for a robot. These serve to protect the system. A maximum of 8 Cartesian and 8 axis-specific workspaces can be configured at any one time. The workspaces can overlap.

Irrespective of whether it is Cartesian or axis-specific, a workspace is always of one of the two following types:

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

Which of the two types a workspace is to belong to is defined during configuration. There are not only 2 types available for selection, however, but several more finely differentiated modes.

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



Workspaces serve merely to protect the system. They are not for protection of personnel. This must be provided using other means. Failure to observe this may result in death, severe injuries or damage to property.

### 6.13.1 Configuring Cartesian workspaces

#### Description

Cartesian workspaces are cuboids. The following parameters define the position and size of a Cartesian workspace:

- Origin and orientation of the workspace relative to the WORLD coordinate system
- Dimensions of the workspace, starting from the origin

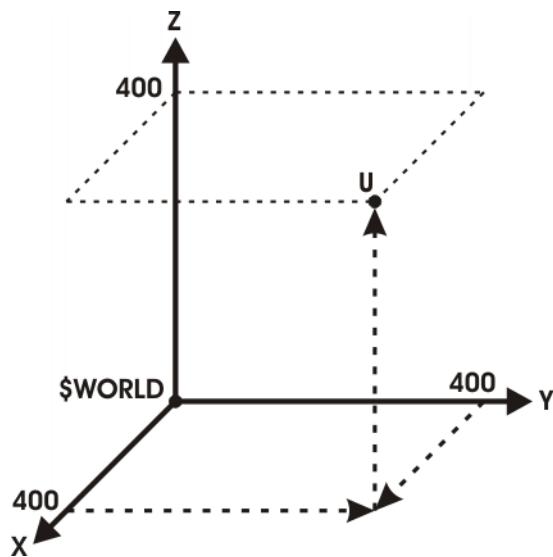


Fig. 6-9: Cartesian workspace, origin U

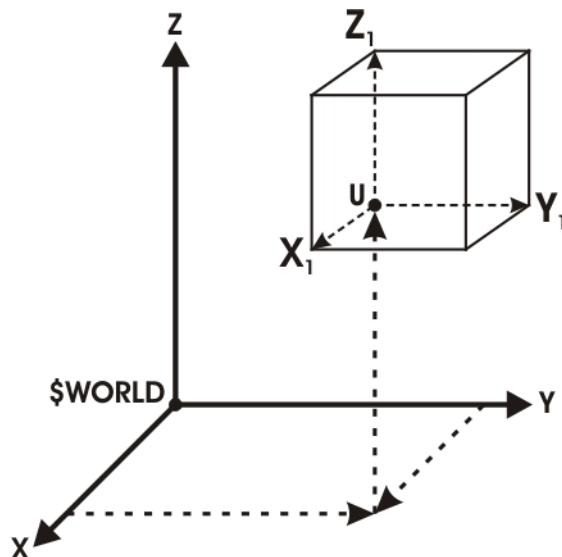


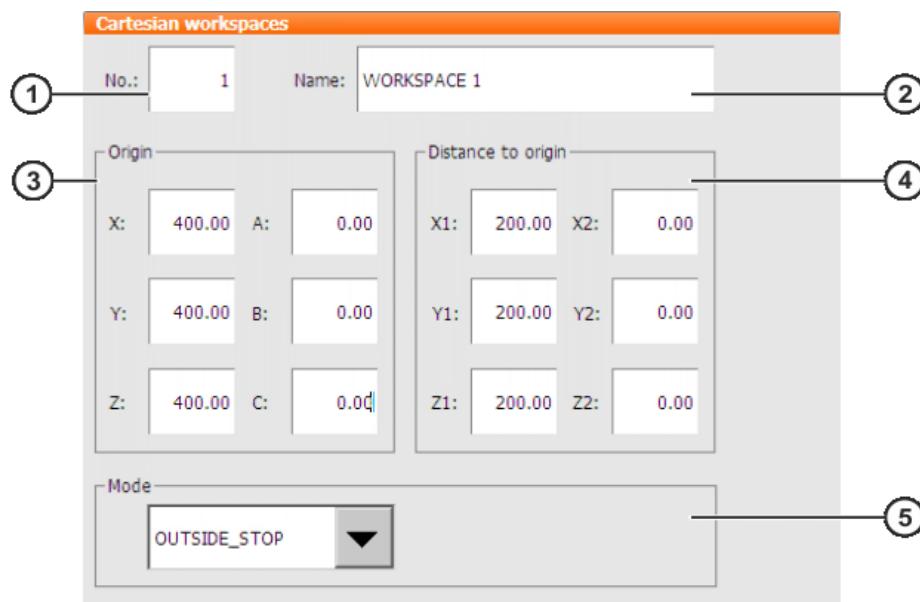
Fig. 6-10: Cartesian workspace, dimensions

#### Precondition

- User group "Expert".
- Operating mode T1 or T2.

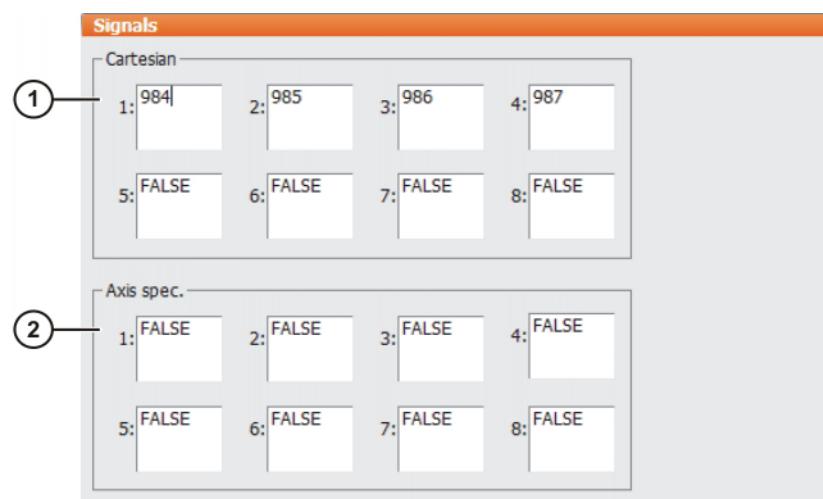
#### Procedure

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.



**Fig. 6-11:** “Cartesian workspaces” window

Item	Description
1	Number of the workspace (max. 8)
2	Designation of the workspace
3	Origin and orientation of the workspace relative to the WORLD coordinate system
4	Dimensions of the workspace in mm
5	Mode ( <a href="#">6.13.3 "Mode for workspaces" Page 182</a> )



**Fig. 6-12:** “Signals” window

Item	Description
1	Per Cartesian workspace: Output that is set if the workspace is violated.
2	Per axis-specific workspace: Output that is set if the workspace is violated.

FALSE means that no output is assigned to this workspace.

### 6.13.2 Configuring axis-specific workspaces

Description	Axis-specific workspaces can be used to further restrict the areas defined by the software limit switches in order to protect the robot, tool or workpiece.
-------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------

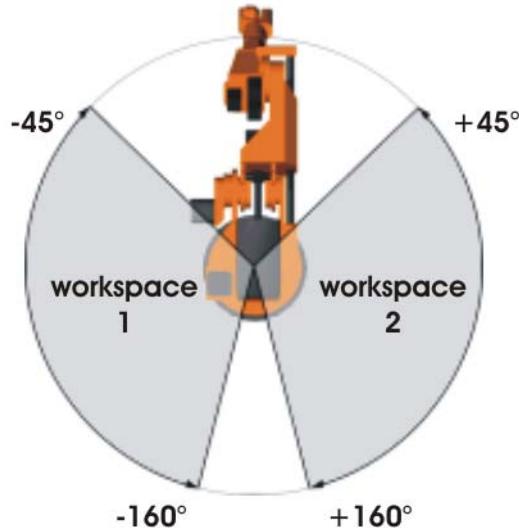


Fig. 6-13: Example of axis-specific workspaces for A1

**Precondition**

- User group "Expert".
- Operating mode T1 or T2.

**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.



Fig. 6-14: “Axis-specific workspaces” window

Item	Description
1	Number of the workspace (max. 8)
2	Designation of the workspace
3	Lower limit for axis angle
4	Upper limit for axis angle
5	Mode ( <a href="#">6.13.3 "Mode for workspaces" Page 182</a> )

If the value 0 is entered for an axis under Item 3 and Item 4, the axis is not monitored, irrespective of the mode.

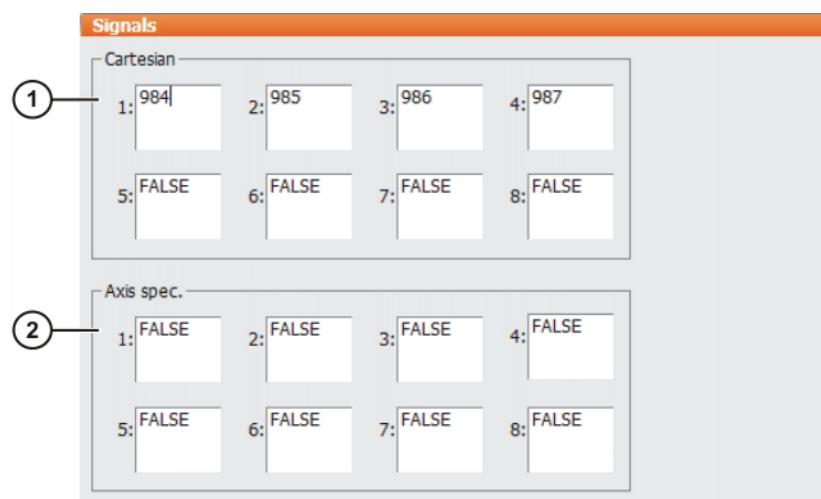


Fig. 6-15: “Signals” window

Item	Description
1	Per Cartesian workspace: Output that is set if the workspace is violated.
2	Per axis-specific workspace: Output that is set if the workspace is violated.

FALSE means that no output is assigned to this workspace.

### 6.13.3 Mode for workspaces

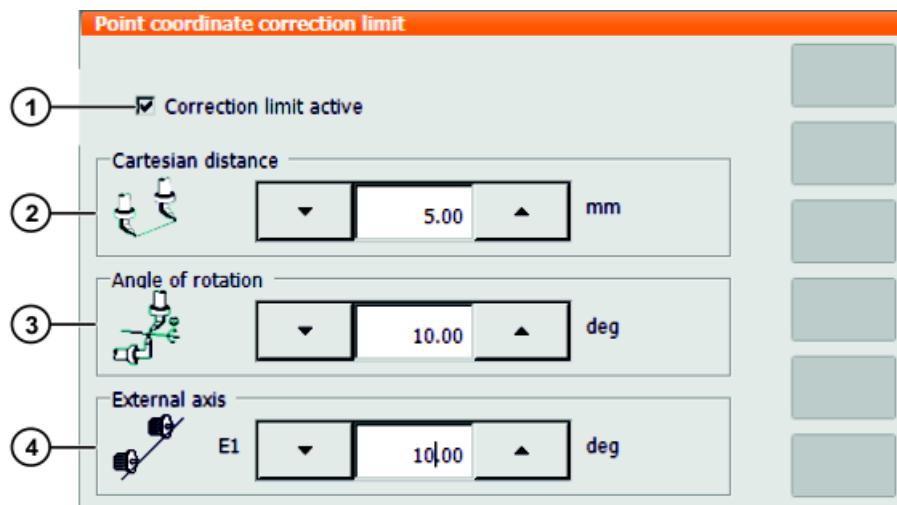
Mode	Description
#OFF	Workspace monitoring is deactivated.
#INSIDE	<ul style="list-style-type: none"> <li>■ Cartesian workspace: The defined output is set if the TCP or flange is located inside the workspace.</li> <li>■ Axis-specific workspace: The defined output is set if the axis is located inside the workspace.</li> </ul>
#OUTSIDE	<ul style="list-style-type: none"> <li>■ Cartesian workspace: The defined output is set if the TCP or flange is located outside the workspace.</li> <li>■ Axis-specific workspace: The defined output is set if the axis is located outside the workspace.</li> </ul>
#INSIDE_STOP	<ul style="list-style-type: none"> <li>■ Cartesian workspace: 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)</li> <li>■ Axis-specific workspace: The defined output is set if the axis is located inside the workspace.</li> </ul> <p>The robot is also stopped and messages are displayed. The robot cannot be moved again until the workspace monitoring is deactivated or bypassed.</p> <p>(&gt;&gt;&gt; 4.17 "Bypassing workspace monitoring" Page 76)</p>
#OUTSIDE_STOP	<ul style="list-style-type: none"> <li>■ Cartesian workspace: The defined output is set if the TCP or flange is located outside the workspace.</li> <li>■ Axis-specific workspace: The defined output is set if the axis is located outside the workspace.</li> </ul> <p>The robot is also stopped and messages are displayed. The robot cannot be moved again until the workspace monitoring is deactivated or bypassed.</p> <p>(&gt;&gt;&gt; 4.17 "Bypassing workspace monitoring" Page 76)</p>

## 6.14 Defining limits for reteaching

<b>Description</b>	If this function is active, existing points may only be re-taught 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. Global positions, e.g. the HOME position, can no longer be re-taught at all in these user groups.  In the user group “Expert”, points can still be re-taught without restrictions.
<b>Precondition</b>	<ul style="list-style-type: none"> <li>■ “Expert” user group</li> </ul>

**Procedure**

1. In the main menu, select **Configuration > Miscellaneous > Point coordinate correction limit**. The **Point coordinate correction limit** window opens.
2. Enter the desired values and activate the check box **Correction limit active**.
3. Touch the **Close** icon. A request for confirmation is displayed, asking if the changes are to be saved.
4. Answer the query with **Yes**. The entries are saved and the window is closed.



**Fig. 6-16: Option window “Point coordinate correction limit”**

Item	Description
1	Check box active: The limits are active. Check box not active: The limits are not active.
2	Maximum permissible change for the X, Y and Z values <b>■ 1.00 ... 100.00 mm</b> The value represents the radius of an imagined sphere about the original point.
3	Maximum permissible change for the A, B and C values <b>■ 0.00 ... 20.00 deg</b> 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. <b>■ 0.00 ... 100.00 mm</b>

Button	Description
<b>External axes</b>	The button is only available if more than one external axis is present. Displays the next external axis.

## 6.15 Warm-up

If a robot is started in low ambient temperatures, this results in increased friction in the gear unit. This can cause the motor current of an axis (or of more than one axis) to reach its maximum value. This stops the robot and the robot controller generates the error message *Regulator limit exceeded <axis number>*.

To avoid this, the motor current can be monitored during the warm-up phase. If a defined value is reached, the robot controller reduces the motion velocity. This, in turn, reduces the motor current.



The monitoring refers to PTP motions and PTP-CP approximate positioning blocks.

Other motions are not monitored and their velocity is not reduced. These include LIN, CIRC and all spline motions (CP and PTP).

### 6.15.1 Warm-up sequence

#### Precondition

- \$WARMUP\_RED\_VEL = TRUE
- Operating mode AUT or AUT EXT
- The robot is considered to be cold. This applies in the following cases:
  - Cold start
  - Or \$COOLDOWN\_TIME has expired.
  - Or \$WARMUP\_RED\_VEL has been set from FALSE to TRUE.

#### Example

Sequence on the basis of the following example values:

```
$WARMUP_RED_VEL = TRUE  
$WARMUP_TIME = 30  
$COOLDOWN_TIME = 120  
$WARMUP_CURR_LIMIT = 95  
$WARMUP_MIN_FAC = 60  
$WARMUP_SLEW_RATE = 5
```

1. The cold robot starts. The motor currents are monitored for 30 minutes (\$WARMUP\_TIME).  
2. If the motor current of an axis exceeds 95% (\$WARMUP\_CURR\_LIMIT) of the maximum permissible motor current, the monitoring is triggered. The robot controller then generates the message *Warm-up active* and reduces the internal override. The robot slows down and the motor current drops. The program override displayed on the user interface remains unchanged.

The internal override is reduced to a maximum of 60% (\$WARMUP\_MIN\_FAC) of the programmed override. There is no way of influencing how quickly the internal override is reduced.

The system variable \$WARMUP\_RED\_FAC\_ACT indicates the percentage of the program override to which the internal override is currently reduced by the warm-up:

- Internal override = \$OV\_PRO \* \$WARMUP\_RED\_FAC\_ACT

Example: A program override \$OV\_PRO of 90% and a reduction factor \$WARMUP\_RED\_FAC\_ACT of 95% result in an internal override of  $90\% \cdot 95\% = 85.4\%$ .

3. Once the monitoring is no longer triggered, the robot controller increases the internal override again. This is generally the case before the minimum \$WARMUP\_MIN\_FAC has been reached. The robot accelerates again. Once a second, the robot controller edges back up towards the programmed override. \$WARMUP\_SLEW\_RATE determines the rate of increase. In the example, the internal override is increased by 5% per second.
4. It is possible that the robot is still not warm enough and that the motor current thus exceeds the maximum \$WARMUP\_CURR\_LIMIT once again. The robot controller reacts (within \$WARMUP\_TIME) the same way as the first time.

5. If the robot is warm enough for the robot controller to increase the internal override all the way back up to the programmed override, the robot controller deactivates the message *Warm-up active*.
6. After 30 minutes (\$WARMUP\_TIME), the robot is deemed to be warmed up and the motor currents are no longer monitored.

**LOG file**

The following events are logged in the file “Warmup.LOG” (path “KRC:\Robot\Log\”):

Entry	Meaning
<b>Monitoring active</b>	The motor currents are monitored.
<b>Monitoring inactive</b>	The motor currents are not monitored.
<b>Controlling active</b>	The velocity is reduced.
<b>Controlling inactive</b>	The velocity corresponds to the programmed override once again.

Example:

```
...
Date: 21.08.08 Time: 14:46:57 State: Monitoring active
Date: 21.08.08 Time: 14:54:06 State: Controlling active
Date: 21.08.08 Time: 14:54:07 State: Controlling inactive
Date: 21.08.08 Time: 18:23:43 State: Monitoring inactive
...
```

### 6.15.2 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	<ul style="list-style-type: none"> <li>■ <b>TRUE</b>: Warm-up function is activated.</li> <li>■ <b>FALSE</b>: Warm-up function is deactivated. (Default)</li> </ul> <p>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.</p>
\$WARMUP_TIME	<p>Time during which the motor currents are monitored by the warm-up function.</p> <p>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.</p> <ul style="list-style-type: none"> <li>■ &gt; 0.0 min</li> </ul>

System variable	Description
\$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. <ul style="list-style-type: none"><li>■ &gt; 0.0 min</li></ul>
\$WARMUP_CURR_LIMIT	Maximum permissible motor current during warm-up (relative to the regular maximum permissible motor current)  Regular maximum permissible motor current = (\$CURR_LIM * \$CURR_MAX) / 100 <ul style="list-style-type: none"><li>■ 0 ... 100%</li></ul>
\$WARMUP_RED_FAC_ACT	Current reduction factor for program override <ul style="list-style-type: none"><li>■ \$WARMUP_MIN_FAC ... 100%</li></ul> 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. <ul style="list-style-type: none"><li>■ 0 ... 100%</li></ul>
\$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. <ul style="list-style-type: none"><li>■ &gt; 0.0%/s</li></ul>

## 6.16 Collision detection



Collision detection is an extension and improvement of the torque monitoring function.

The torque monitoring function remains available. Programs in which torque monitoring is already defined can still be executed as before. Alternatively, the lines with the torque monitoring in such programs can be deleted and collision detection can be used instead. Collision detection must not be used together with torque monitoring in a program.

### Description

If the robot collides with an object, the robot controller increases the axis torques in order to overcome the resistance. This can result in damage to the robot, tool or other objects.

Collision detection reduces the risk of such damage. It monitors the axis torques. If these deviate from a specified tolerance range, the following reactions are triggered:

- The robot stops with a STOP 1.
  - The robot controller calls the program **tm\_useraction**. This is located in the **Program** folder and contains the HALT statement. Alternatively, the user can program other reactions in the program **tm\_useraction**.
- (>>>> 6.16.4 "Editing the program tm\_useraction" Page 191)

The robot controller automatically calculates the tolerance range. (Exception: no values are calculated in T1 mode.) A program must generally be executed 2 or 3 times before the robot controller has calculated a practicable tolerance range. The user can define an offset via the user interface for the tolerance range calculated by the robot controller.

If the robot is not operated for a longer period (e.g. over the weekend), the motors, gear units, etc., cool down. Different axis torques are required in the first few runs after such a break than in the case of a robot that is already at operating temperature. The robot controller automatically adapts the collision detection to the changed temperature.

#### Precondition

- In order to be able to use the collision detection function, acceleration adaptation must be activated.  
Acceleration adaptation is activated when system variable \$ADAP\_ACC is not equal to #NONE. (This is the default setting.) The system variable can be found in the file C:\KRC\Roboter\KRC\R1\MaDa\\$ROBCOR.DAT.
- The tolerance range is only calculated for motion blocks that have been executed completely.
- To activate collision detection for a motion, the parameter **Collision detection** must be set to TRUE during programming. This can be seen from the addition CD in the program code:

```
PTP P2 Vel= 100 % PDAT1 Tool[1] Base[1] CD
```



The parameter **Collision detection** is only available if the motion is programmed via an inline form.  
Information about collision detection for motions programmed without inline forms can be found in the documentation "KUKA.ExpertTech".

#### Limitations

- Collision detection is not possible for HOME positions and other global positions.
- Collision detection is not possible for external axes.
- Collision detection is not possible during backward motion.
- Collision detection is not possible in T1 mode.
- High axis torques arise when the stationary robot starts to move. For this reason, the axis torques are not monitored in the starting phase (approx. 700 ms).
- The collision detection function reacts much less sensitively for the first 2 or 3 program executions after the program override value has been modified. Thereafter, the robot controller has adapted the tolerance range to the new program override.

#### System variables

System variable	Description
\$TORQ_DIFF	The values of \$TORQ_DIFF (torque) and \$TORQ_DIFF2 (impact) are automatically calculated during program execution. These values are compared with the values from the previous program execution or with the default values. The highest value is saved. The values are always calculated, even when collision detection is deactivated.
\$TORQ_DIFF2	If collision detection is active, the system compares the values of \$TORQ_DIFF and \$TORQ_DIFF2 with the saved values during the motion.
\$TORQMON_DEF[1] ... \$TORQMON_DEF[6]	Values for the tolerance range in program mode (per axis)* File KRC:\STEU\Mada\\$custom.dat
\$TORQMON_COM_DEF[1] ... \$TORQMON_COM_DEF[6]	Values for the tolerance range in jog mode (per axis)* File KRC:\STEU\Mada\\$custom.dat
\$COLL_ENABLE	Signal declaration. This output is set if the value of one of the \$TORQMON_DEF[...] variables is less than 200. File: KRC:\STEU\Mada\\$machine.dat
\$COLL_ALARM	Signal declaration. This output is set if message 117 "Collision Detection axis <axis number>" is generated. The output remains set as long as \$STOPMESS is active. File: KRC:\STEU\Mada\\$machine.dat
\$TORQMON_TIME	Response time for the collision detection. Unit: milliseconds. Default value: 0.0 File: C:\KRC\Roboter\KRC\Steu\MaDa\\$CUSTOM.DAT.

\*The width of the tolerance range is equal to the maximum torque [Nm] multiplied by the value in \$TORQMON\_... . The default value is 200. Unit: percent.

### 6.16.1 Calculating the tolerance range and activating collision detection

#### Precondition

- The acceleration adaptation is switched on.
- The load data have been entered correctly.
- In the program, the parameter **Collision detection** is set to TRUE for all motions that are to be monitored.
- If required: the desired collision response has been programmed in the program **tm\_useraction**.

#### Procedure

1. In the main menu, select **Configuration > Miscellaneous > Collision detection**.  
(>>> 6.16.3 "Option window "Collision detection"" Page 190)
2. The box **KCP** must contain the entry **MonOff**. If this is not the case, press **Deactivate**.
3. Start the program and execute it several times. After 2 or 3 program executions, the robot controller has calculated a practicable tolerance range.
4. Press **Activate**. The box **KCP** in the **Collision detection** window now contains the entry **MonOn**.  
Save the configuration by pressing **Close**.

If required, the user can define an offset for the tolerance range.  
(>>> 6.16.2 "Defining an offset for the tolerance range" Page 189)



The tolerance range must be recalculated in the following cases:

- The velocity has been modified.
- Points have been changed, added or removed.

## 6.16.2 Defining an offset for the tolerance range

**Description** An offset for the torque and for the impact can be defined for the tolerance range. The lower the offset, the more sensitive the reaction of the collision detection. The higher the offset, the less sensitive the reaction of the collision detection.

**Torque:** The torque is effective if the robot meets a continuous resistance. Examples:

- The robot collides with a wall and pushes against the wall.
- The robot collides with a container. The robot pushes against the container and moves it.

**Impact:** The impact is effective if the robot meets a brief resistance. Example:

- The robot collides with a panel which is sent flying by the impact.



If the collision detection reacts too sensitively, do not immediately increase the offset. Instead, recalculate the tolerance range first and test whether the collision detection now reacts as desired.

(>>> 6.16.1 "Calculating the tolerance range and activating collision detection" Page 188)

**Procedure**

1. Select program.
2. In the main menu, select **Configuration > Miscellaneous > Collision detection**.  
(>>> 6.16.3 "Option window "Collision detection"" Page 190)
3. The offset for a motion can be modified while a program is running:  
If the desired motion is displayed in the **Collision detection** window, press an arrow key in the **Torque** or **Impact** box. The window remains focused on the motion and the offset can be modified.  
Alternatively, a block selection to the desired motion can be carried out.
4. Save the change by pressing **Save**.
5. Save the configuration by pressing **Close**.
6. Set the original operating mode and program run mode.

### 6.16.3 Option window “Collision detection”

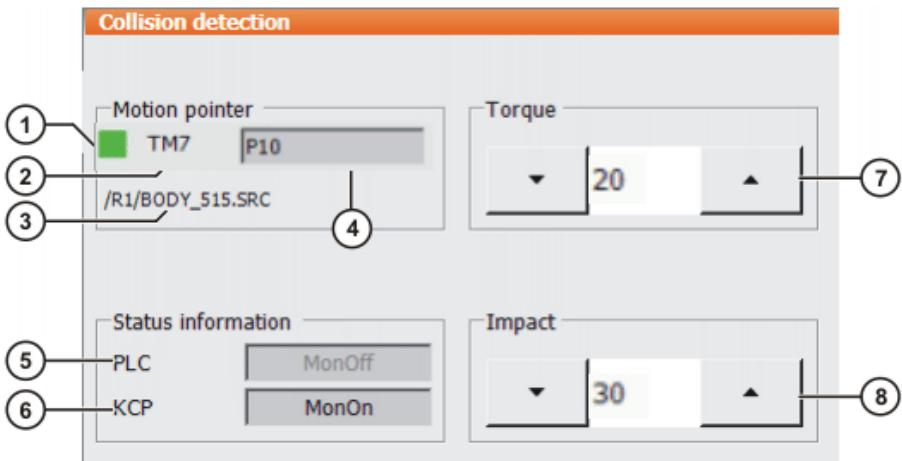


Fig. 6-17: Option window “Collision detection”



The values in the option window **Collision detection** do not always refer to the current motion. Deviations are particularly possible in the case of points which are close together and approximated motions.

Item	Description
1	Indicates the status of the current motion: <ul style="list-style-type: none"> <li>■ Red: the current motion is not monitored.</li> <li>■ Green: the current motion is monitored.</li> <li>■ Orange: an arrow key has been pressed in the <b>Torque</b> or <b>Impact</b> box. The window remains focused on the motion and the offset can be modified. The change can then be applied by pressing <b>Save</b>.</li> <li>■ Pixelated: A program must generally be executed 2 or 3 times before the robot controller has calculated a practicable tolerance range. This display is pixelated during this learning phase.</li> </ul>
2	Number of the TMx variable  The robot controller creates a TMx variable for each motion block in which the parameter <b>Collision detection</b> is set to TRUE. TMx contains all the values for the tolerance range of this motion block. If 2 motion blocks refer to the same point Px, the robot controller creates 2 TMx variables.
3	Path and name of the selected program
4	Point name
5	This box is only active in “Automatic External” mode. It appears gray in all other modes.  <b>MonOn:</b> collision detection has been activated by the PLC.  If collision detection is activated by the PLC, the PLC sends the input signal <b>sTQM_SPSACTIVE</b> to the robot controller. The robot controller responds with the output signal <b>sTQM_SPSSTATUS</b> . The signals are defined in the file \$config.dat.  <b>Note:</b> Collision detection is only active in Automatic External mode if both the <b>PLC</b> box and the <b>KCP</b> box show the entry <b>MonOn</b> .

Item	Description
6	<p><b>MonOn:</b> collision detection has been activated from the smart-PAD.</p> <p><b>Note:</b> Collision detection is only active in Automatic External mode if both the <b>PLC</b> box and the <b>KCP</b> box show the entry <b>MonOn</b>.</p>
7	<p>Offset for the torque. The lower the offset, the more sensitive the reaction of the collision detection. Default value: 20.</p> <p>If an arrow key is pressed, the window remains focused on the motion and the offset can be modified. The change can then be applied by pressing <b>Save</b>.</p> <ul style="list-style-type: none"> <li>■ <b>N.A.:</b> the option <b>Collision detection</b> in the inline form is set to FALSE for this motion.</li> </ul>
8	<p>Offset for the impact. The lower the offset, the more sensitive the reaction of the collision detection. Default value: 30.</p> <p>If an arrow key is pressed, the window remains focused on the motion and the offset can be modified. The change can then be applied by pressing <b>Save</b>.</p> <ul style="list-style-type: none"> <li>■ <b>N.A.:</b> the option <b>Collision detection</b> in the inline form is set to FALSE for this motion.</li> </ul>

Button	Description
<b>Activate</b>	<p>Activates collision detection.</p> <p>This button is not displayed if the torque or impact has been changed, but the changes have not yet been saved.</p>
<b>Deactivate</b>	<p>Deactivates collision detection.</p> <p>This button is not displayed if the torque or impact has been changed, but the changes have not yet been saved.</p>
<b>Save</b>	Saves changes to the torque and/or impact.
<b>Cancel</b>	Rejects changes to the torque and/or impact.

#### 6.16.4 Editing the program tm\_useraction

- Description** By default, the program **tm\_useraction** contains the HALT statement. If required, the user can program other statements.
- Preparation**
- In the attributes of the program **tm\_useraction**, activate the attribute **Visible**:  
To do so, activate the check box **Visible** on the **Module info** tab.  
([">>>> 7.4.2 "Displaying or modifying properties of files and folders"](#)  
Page 239)
- Precondition**
- “Expert” user group
  - T1, T2 or AUT mode
  - Submit interpreter is deselected.
- Procedure**
1. Select **tm\_useraction.srC** in the file list (= right-hand area of the Navigator).
  2. Press the **Open** button. The program is displayed in the editor.
  3. Make the desired changes.



**tm\_useraction.scr** is called by the robot controller by means of an interrupt. The restrictions that apply to interrupt programs must therefore be observed during programming.

4. Close the program.

To accept the changes, answer the request for confirmation with **Yes**.

5. Recommendation: Deactivate the attribute **Visible** again.

Reason:

- **Visible** activated: when the robot controller calls **tm\_useraction**, the block pointer indicates this program.
- **Visible** deactivated: when the robot controller calls **tm\_useraction**, the block pointer indicates the point at which the main program was interrupted. This is generally more useful during troubleshooting.

#### 6.16.5 Torque monitoring



Collision detection is an extension and improvement of the torque monitoring function.

The torque monitoring function remains available. Programs in which torque monitoring is already defined can still be executed as before. Alternatively, the lines with the torque monitoring in such programs can be deleted and collision detection can be used instead. Collision detection must not be used together with torque monitoring in a program.

##### Description

Differences between torque monitoring and collision detection:

- The tolerance range is not automatically calculated by the robot controller, but must be defined by the user.
- The tolerance range only refers to the torque. No values can be defined for the impact.
- The robot controller cannot automatically adapt the tolerance range to changed temperatures.
- If a collision is detected, the robot stops with a STOP 1. It is not possible to call a user-defined program.

##### Overview

Step	Description
1	Determine suitable values for torque monitoring.  (>>> 6.16.5.1 "Determining values for torque monitoring" Page 192)
2	Program torque monitoring.  (>>> 6.16.5.2 "Programming torque monitoring" Page 193)

##### 6.16.5.1 Determining values for torque monitoring

##### Description

The maximum torque deviation that has occurred can be determined as a percentage by means of the system variable \$TORQ\_DIFF[...].

##### Procedure

1. In the main menu, select **Display > Variable > Single**.
2. Set the value of the variable \$TORQ\_DIFF[...] to 0.
3. Execute the motion block and read the variable again. The value corresponds to the maximum torque deviation.
4. Set the variable for the monitoring of the axis to this value plus a safety margin of 5 - 10%.

Only the value 0 can be assigned to the variables \$TORQ\_DIFF[...].

### 6.16.5.2 Programming torque monitoring

#### Precondition

- In order to be able to use the collision detection function, acceleration adaptation must be activated. Acceleration adaptation is activated when system variable \$ADAP\_ACC is not equal to #NONE. (This is the default setting.) The system variable can be found in the file C:\KRC\Robot-er\KRCR1\MaDa\\$ROBCOR.DAT.
- A program is selected.

#### Procedure

1. Position the cursor in the line before the motion for which the torque monitoring is to be programmed.
2. Select the menu sequence **Commands > Motion parameters > Torque monitoring**. An inline form is opened.

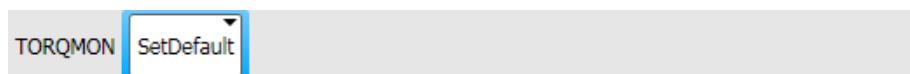


Fig. 6-18: Inline form “TORQMON SetDefault”

3. In the **TORQMON** box, select the entry **SetLimits**.



Fig. 6-19: Inline form “TORQMON SetLimits”

4. For each axis, enter the amount by which the command torque may deviate from the actual torque.
5. Press **Cmd OK**.
6. If a response time for the torque monitoring is to be defined:  
Set the variable \$TORQMON\_TIME to the desired value. Unit: milliseconds. Default value: 0.

The values are automatically reset to the default value 200 in the following cases:

- Reset
- Block selection
- Program deselection

## 6.17 Defining calibration tolerances



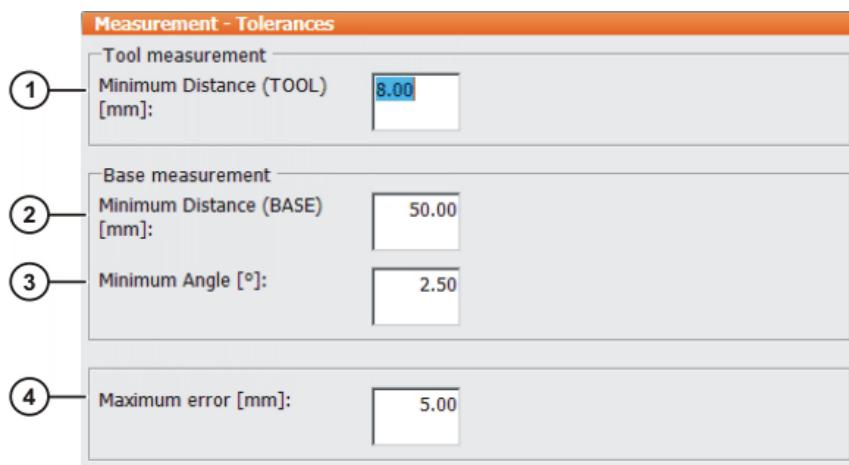
Only modify the default values in exceptional cases. Otherwise, increased error messages and inaccuracy may result.

#### Precondition

- “Expert” user group

#### Procedure

- In the main menu, select **Start-up > Calibrate > Tolerances**.

**Description****Fig. 6-20: 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
<b>Default</b>	Restores the default settings. The data must then be saved by pressing <b>OK</b> .

**6.18 Configuring backward motion****Description**

The configuration options described below apply to backward motion using the Start backwards key. They do not apply to other backward motion functionalities, e.g. backward motion as part of fault strategies in technology packages.

The following default settings are valid for backward motion using the Start backwards key:

- Backward motion is active.
- A maximum of 30 motions can be saved in the buffer.
- When backward motion is started, the robot controller does not indicate this by means of a message.

If other settings are desired, these must be entered in KrcConfig.XML.

**Precondition**

- User group "Expert"
- KSS: T1, T2 or AUT mode
- VSS: T1 or T2 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" MOUEMENTS="20"
  ↴ BACKWARD_WARNING="true"/>

178  </KrcConfig>

```

**Fig. 6-21: 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 <ul style="list-style-type: none"> <li>■ <b>TRUE</b> (= default): activated, i.e. backward motion using the Start backwards key is possible.</li> <li>■ <b>FALSE</b>: deactivated, i.e. backward motion using the Start backwards key is not possible.</li> </ul>
MOVE-MENTS	Type: INT <p>Maximum number of motions recorded for backward motion</p> <ul style="list-style-type: none"> <li>■ <b>0 ... 60</b> (default = 30)</li> </ul>
BACKWARD_WARNING	Type: BOOL <ul style="list-style-type: none"> <li>■ <b>TRUE</b>: The robot controller generates the following message when the Start backwards key is pressed for the first time after forward motion: <i>Caution! Robot is moving backwards..</i> The user must acknowledge the message and press the Start backwards key again.</li> <li>■ <b>FALSE</b> (= default): No message</li> </ul>

## 6.19 Configuring Automatic External

<b>Description</b>	If robot processes are to be controlled centrally by a higher-level controller (e.g. a PLC), this is carried out using the Automatic External interface.  The higher-level controller transmits the signals for the robot processes (e.g. fault acknowledgment, program start, etc.) to the robot controller via the Automatic External interface. The robot controller transmits information about operating states and fault states to the higher-level controller.
--------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Overview**

To enable use of the Automatic External interface, the following configurations must be carried out:

Step	Description
1	Configuration of the CELL.SRC program. (>>> 6.19.1 "Configuring CELL.SRC" Page 196)
2	Configuration of the inputs/outputs of the Automatic External interface. (>>> 6.19.2 "Configuring Automatic External inputs/outputs" Page 197)
3	Only if error numbers are to be transmitted to the higher-level controller: configuration of the P00.DAT file. (>>> 6.19.3 "Transmitting error numbers to the higher-level controller" Page 204)

**6.19.1 Configuring CELL.SRC**

**Description** In Automatic External mode, programs are called using the program CELL.SRC.

**Program**

```

1  DEF    CELL ( )
...
6   INIT
7   BASISTECHINI
8   CHECK HOME
9   PTP HOME  Vel= 100 % DEFAULT
10  AUTOEXTINI
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.

<b>Line</b>	<b>Description</b>
27	DEFAULT = the program number is invalid.
28	Error treatment in the case of an invalid program number

- Precondition** ■ “Expert” user group

- Procedure**
1. Open the program CELL.SRC in the Navigator. (This program is located in the folder "R1".)
  2. In the section CASE 1, replace the name EXAMPLE1 with the name of the program that is to be called via program number 1. Delete the semicolon in front of the name.

```
...
15      CASE 1
16          P00 (#EXT_PGNO,#PGNO_ACKN,DMY[],0 )
17          MY_PROGRAM ( ) ; Call User-Program
...
...
```

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

### 6.19.2 Configuring Automatic External inputs/outputs

- Procedure**
1. In the main menu, select **Configuration > Inputs/outputs > Automatic External**.
  2. In the **Value** column, select the cell to be edited and press **Edit**.
  3. Enter the desired value and save it by pressing **OK**.
  4. Repeat steps 2 and 3 for all values to be edited.
  5. Close the window. The changes are saved.

**Description**

Term	Type	Name	Value
1 Type programno.	Var	PGNO_TYPE	1
2 programno. reflection	Var	REFLECT_PROG_NR	0
3 Bitwidth programno.	Var	PGNO_LENGTH	8
4 First bit programno.	IO	PGNO_FBIT	33
5 Parity bit	IO	PGNO_PARITY	41
6 Programno. valid	IO	PGNO_VALID	42
7 Programstart	IO	\$EXT_START	1026
8 Move enable	IO	\$MOVE_ENABLE	1025
9 Error confirmation	IO	\$CONF_MESS	1026
10 Drives off (invers)	IO	\$DRIVES_OFF	1025
11 Drives on	IO	\$DRIVES_ON	140
12 Activate interface	IO	\$I_O_ACT	1025

Fig. 6-22: Configuring Automatic External inputs

Term	Type	Name	Value
1 Control ready	IO	\$RC_RDY1	137
2 Alarm stop active	IO	\$ALARM_STOP	1013
3 User safety switch closed	IO	\$USER_SAF	1011
4 Drives ready	IO	\$PERI_RDY	1012
5 Robot calibrated	IO	\$ROB_CAL	1001
6 Interface active	IO	\$I_O_ACTCONF	140
7 Error collection	IO	\$STOPMESS	1010
8 First bit for programno. reflection	IO	PGNO_FBIT_REFL	999
9 Internal emergency stop	IO	IntEstop	1002

Fig. 6-23: Configuring Automatic External outputs

Item	Description
1	Number
2	Long text name of the input/output

Item	Description
3	Type <ul style="list-style-type: none"> <li>■ <b>Green:</b> Input/output</li> <li>■ <b>Yellow:</b> variable or system variable (\$...)</li> </ul>
4	Name of the signal or variable
5	Input/output number or channel number
6	The outputs are thematically assigned to tabs.

#### 6.19.2.1 Automatic External inputs

**PGNO\_TYPE** Type: Variable

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

Value	Description	Example
1	Read as binary number. The program number is transmitted by the higher-level controller as a binary coded integer.	0 0 1 0 0 1 1 1 => PGNO = 39
2	Read as BCD value. The program number is transmitted by the higher-level controller as a binary coded decimal.	0 0 1 0 0 1 1 1 => PGNO = 27
3	Read as "1 of n". The program number is transmitted by the higher-level controller or the periphery as a "1 of n" coded value.	0 0 0 0 0 0 0 1 => PGNO = 1 0 0 0 0 1 0 0 0 => PGNO = 4

\* When using this transmission format, the values of PGNO\_REQ, PGNO\_PARITY and PGNO\_VALID are not evaluated and are thus of no significance.

**REFLECT\_PROG** Type: Variable

**\_NR** This variable defines whether the program number is to be mirrored to an output range. The output of the signal starts with the output defined using PGNO\_FBIT\_REFL.

Value	Description
0	Function deactivated
1	Function activated

**PGNO\_LENGTH** Type: Variable

This variable determines the number of bits in the program number sent by the higher-level controller. Range of values: 1 ... 16.

Example: PGNO\_LENGTH = 4 => the external program number is 4 bits long.

If PGNO\_TYPE has the value 2, only 4, 8, 12 and 16 are permissible values for the number of bits.

**PGNO\_FBIT** Input representing the first bit of the program number. Range of values: 1 ... 8192.

Example: PGNO\_FBIT = 5 => the external program number begins with the input \$IN[5].

#### **PGNO\_PARITY**

Input to which the parity bit is transferred from the higher-level controller.

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

(>>> 6.19.2.2 "Odd / even parity" Page 201)

If PGNO\_TYPE has the value 3, PGNO\_PARITY is not evaluated.

#### **PGNO\_VALID**

Input to which the command to read the program number is transferred from the higher-level controller.

Input	Function
Negative value	Number is transferred at the falling edge of the signal.
0	Number is transferred at the rising edge of the signal on the EXT_START line.
Positive value	Number is transferred at the rising edge of the signal.

If PGNO\_TYPE has the value 3, PGNO\_VALID is not evaluated.

#### **\$EXT\_START**

If the I/O interface is active, this input can be set to start or continue a program.



Only the rising edge of the signal is evaluated.

#### **NOTICE**

There is no BCO run in Automatic External mode. This means that the robot moves to the first programmed position after the start at the programmed (not reduced) velocity and does not stop there.

#### **\$MOVE\_ENABLE**

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

Signal	Function
TRUE	Jogging and program execution are possible.
FALSE	All drives are stopped and all active commands inhibited.

If the drives have been switched off by the higher-level controller, the message "GENERAL MOTION ENABLE" is displayed. It is only possible to move the robot again once this message has been reset and another external start signal has been given.

During commissioning, the variable \$MOVE\_ENABLE is often configured with the value \$IN[1025]. If a different input is not subsequently configured, no external start is possible.

#### **\$CHCK\_MOVENA**

Type: Variable

If the variable \$CHCK\_MOVENA has the value FALSE, \$MOVE\_ENABLE can be bypassed. The value of the variable can only be changed in the file C:\KRC\ROBOTER\KRC\STEU\Mada\\$OPTION.DAT.

Signal	Function
TRUE	MOVE_ENABLE monitoring is activated.
FALSE	MOVE_ENABLE monitoring is deactivated.



In order to be able to use MOVE\_ENABLE monitoring, \$MOVE\_ENABLE must have been configured with the input \$IN[1025]. Otherwise, \$CHCK\_MOVENA has no effect.

#### \$CONF\_MESS

Setting this input enables the higher-level controller to acknowledge error messages automatically as soon as the cause of the error has been eliminated.



Only the rising edge of the signal is evaluated.

#### \$DRIVES\_OFF

If there is a low-level pulse of at least 20 ms duration at this input, the higher-level controller switches off the robot drives.

#### \$DRIVES\_ON

If there is a high-level pulse of at least 20 ms duration at this input, the higher-level controller switches on the robot drives.

#### \$I\_O\_ACT

If this input is TRUE, the Automatic External interface is active. Default setting: \$IN[1025].

### 6.19.2.2 Odd / even parity

#### Description

A parity bit is a bit which is added to a bit sequence and has a checking function. It indicates whether the bit sequence contains an even or odd sum of ones.

If the entire data block – consisting of the bit sequence and the parity bit – is transferred and the parity bit then no longer matches the sequence, this means that an error has occurred during transfer.

The meaning of the relevant bit value (“even” or “odd”) depends on the applicable parity protocol: “even parity” or “odd parity”.

#### Even parity

The sum of the ones in the entire data block (bit sequence + parity bit) must be even.

Sum of ones in the bit sequence	Parity bit
Even	0
Odd	1

Example:

Bit sequence	Parity bit
0011.1010	0
1010.0100	1

#### Odd parity

#### Odd parity

The sum of the ones in the entire data block (bit sequence + parity bit) must be odd.

Sum of ones in the bit sequence	Parity bit
Even	1
Odd	0

Example:

Bit sequence	Parity bit
0011.1010	1
1010.0100	0

### 6.19.2.3 Automatic External outputs

<b>\$RC_RDY1</b>	Ready for program start.
<b>\$ALARM_STOP</b>	<p>This output is reset in the following EMERGENCY STOP situations:</p> <ul style="list-style-type: none"> <li>■ The EMERGENCY STOP device on the smartPAD is pressed.</li> <li>■ External EMERGENCY STOP</li> </ul> <div style="border: 1px solid black; padding: 10px; margin-top: 10px;">  In the case of an EMERGENCY STOP, the nature of the EMERGENCY STOP can be recognized from the states of the outputs <b>\$ALARM_STOP</b> and <b>\$ALARM_STOP_INTERN</b>:           <ul style="list-style-type: none"> <li>■ Both outputs are FALSE: the EMERGENCY STOP was triggered on the smartPAD.</li> <li>■ <b>\$ALARM_STOP</b> is FALSE, <b>\$ALARM_STOP_INTERN</b> is TRUE: external EMERGENCY STOP.</li> </ul> </div>
<b>\$USER_SAF</b>	This output is reset if the safety fence monitoring switch is opened (AUT mode) or an enabling switch is released (T1 or T2 mode).
<b>\$PERI_RDY</b>	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.
<b>\$ROB_CAL</b>	The signal is FALSE as soon as a robot axis has been unmastered
<b>\$I_O_ACTCONF</b>	This output is TRUE if Automatic External mode is selected and the input <b>\$I_O_ACT</b> is TRUE.
<b>\$STOPMESS</b>	This output is set by the robot controller in order to communicate to the higher-level controller any message occurring which requires the robot to be stopped. (Examples: EMERGENCY STOP, Motion enable or Operator safety)
<b>PGNO_FBIT_REFL</b>	<p>Output representing the first bit of the program number. Precondition: The input REFLECT_PROG_NR has the value 1.</p> <p>The size of the output area depends on the number of bits defining the program number (PGNO_LENGTH).</p> <p>If a program selected by the PLC is deselected by the user, the output area starting with PGNO_FBIT_REFL is set to FALSE. In this way, the PLC can prevent a program from being restarted manually.</p> <p>PGNO_FBIT_REFL is also set to FALSE if the interpreter is situated in the CELL program.</p>
<b>\$ALARM_STOP_INTERN</b>	<p>Previous name: <b>Int. E-Stop</b></p> <p>This output is set to FALSE if the EMERGENCY STOP device on the smartPAD is pressed.</p>

 In the case of an EMERGENCY STOP, the nature of the EMERGENCY STOP can be recognized from the states of the outputs **\$ALARM\_STOP** and **\$ALARM\_STOP\_INTERN**:

- Both outputs are FALSE: the EMERGENCY STOP was triggered on the smartPAD.
- **\$ALARM\_STOP** is FALSE, **\$ALARM\_STOP\_INTERN** is TRUE: external EMERGENCY STOP.

**\$PRO\_ACT**

This output is set whenever a process is active at robot level. The process is therefore active as long as a program or an interrupt is being processed. Program processing is set to the inactive state at the end of the program only after all pulse outputs and all triggers have been processed.

In the event of an error stop, a distinction must be made between the following possibilities:

- If interrupts have been activated but not processed at the time of the error stop, the process is regarded as inactive (**\$PRO\_ACT=FALSE**).
- If interrupts have been activated and processed at the time of the error stop, the process is regarded as active (**\$PRO\_ACT=TRUE**) until the interrupt program is completed or a STOP occurs in it (**\$PRO\_ACT=FALSE**).
- If interrupts have been activated and a STOP occurs in the program, the process is regarded as inactive (**\$PRO\_ACT=FALSE**). If, after this, an interrupt condition is met, the process is regarded as active (**\$PRO\_ACT=TRUE**) until the interrupt program is completed or a STOP occurs in it (**\$PRO\_ACT=FALSE**).

**PGNO\_REQ**

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

If PGNO\_TYPE has the value 3, PGNO\_REQ is not evaluated.

**APPL\_RUN**

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

**\$PRO\_MOVE**

Means that a synchronous axis is moving, including in jog mode. The signal is thus the inverse of **\$ROB\_STOPPED**.

**\$IN\_HOME**

This output communicates to the higher-level controller whether or not the robot is in its HOME position.

**\$ON\_PATH**

This output remains set as long as the robot stays on its programmed path. The output ON\_PATH is set after the BCO run. This output remains set until the robot leaves the path, the program is reset or block selection is carried out. The ON\_PATH signal has no tolerance window, however; as soon as the robot leaves the path the signal is reset.

**\$NEAR\_POSRET**

This signal allows the higher-level controller to determine whether or not the robot is situated within a sphere about the position saved in **\$POS\_RET**. The higher-level controller can use this information to decide whether or not the program may be restarted.

The user can define the radius of the sphere in the file **\$CUSTOM.DAT** using the system variable **\$NEARPATHTOL**.

**\$ROB\_STOPPED**

The signal is set when the robot is at a standstill. In the event of a WAIT statement, this output is set during the wait.

The signal is thus the inverse of **\$PRO\_MOVE**.

**\$T1, \$T2, \$AUT, \$EXT** These outputs are set when the corresponding operating mode is selected.

### 6.19.3 Transmitting error numbers to the higher-level controller

Error numbers of the robot controller in the range 1 to 255 can be transmitted to the higher-level controller. To transmit the error numbers, the file P00.DAT, in the directory C:\KRC\ROBOTER\KRC\R1\TP, must be configured as follows:

```

1 DEFDAT P00
2
3 BOOL PLC_ENABLE=TRUE ; Enable error-code transmission to plc
4 INT I
5 INT F_NO=1
6 INT MAXERR_C=1 ; maximum messages for $STOPMESS
7 INT MAXERR_A=1 ; maximum messages for APPLICATION
8 DECL STOPMESS MLD
9 SIGNAL ERR $OUT[25] TO $OUT[32]
10 BOOL FOUND
11
12 STRUC PRESET INT OUT,CHAR PKG[3],INT ERR
13 DECL PRESET P[255]
...
26 P[1]={OUT 2,PKG[] "P00",ERR 10}
...
30 P[128]={OUT 128,PKG[] "CTL",ERR 1}
...
35 STRUC ERR_MESS CHAR P[3],INT E
36 DECL ERR_MESS ERR_FILE[64]
37 ERR_FILE[1]={P[] "XXX",E 0}
...
96 ERR_FILE[64]={P[] "XXX",E 0}
97 ENDDAT

```

Line	Description
3	PLC_ENABLE must be TRUE.
6	Enter the number of controller errors for the transmission of which parameters are to be defined.
7	Enter the number of application errors for the transmission of which parameters are to be defined.
9	Specify which robot controller outputs the higher-level controller should use to read the error numbers. There must be 8 outputs.
13	In the following section, enter the parameters of the errors. P[1] ... P[127]: range for application errors P[128] ... P[255]: range for controller errors
26	Example of parameters for application errors: <ul style="list-style-type: none"> <li>■ OUT 2 = error number 2</li> <li>■ PKG[] "P00" = technology package</li> <li>■ ERR 10 = error number in the selected technology package</li> </ul>

Line	Description
30	Example of parameters for controller errors: <ul style="list-style-type: none"><li>■ OUT 128 = error number 128</li><li>■ PKG[] "CTL" = technology package</li><li>■ ERR 1 = error number in the selected technology package</li></ul>
37 ... 96	The last 64 errors that have occurred are stored in the ERR_FILE memory.

#### 6.19.4 Signal diagrams

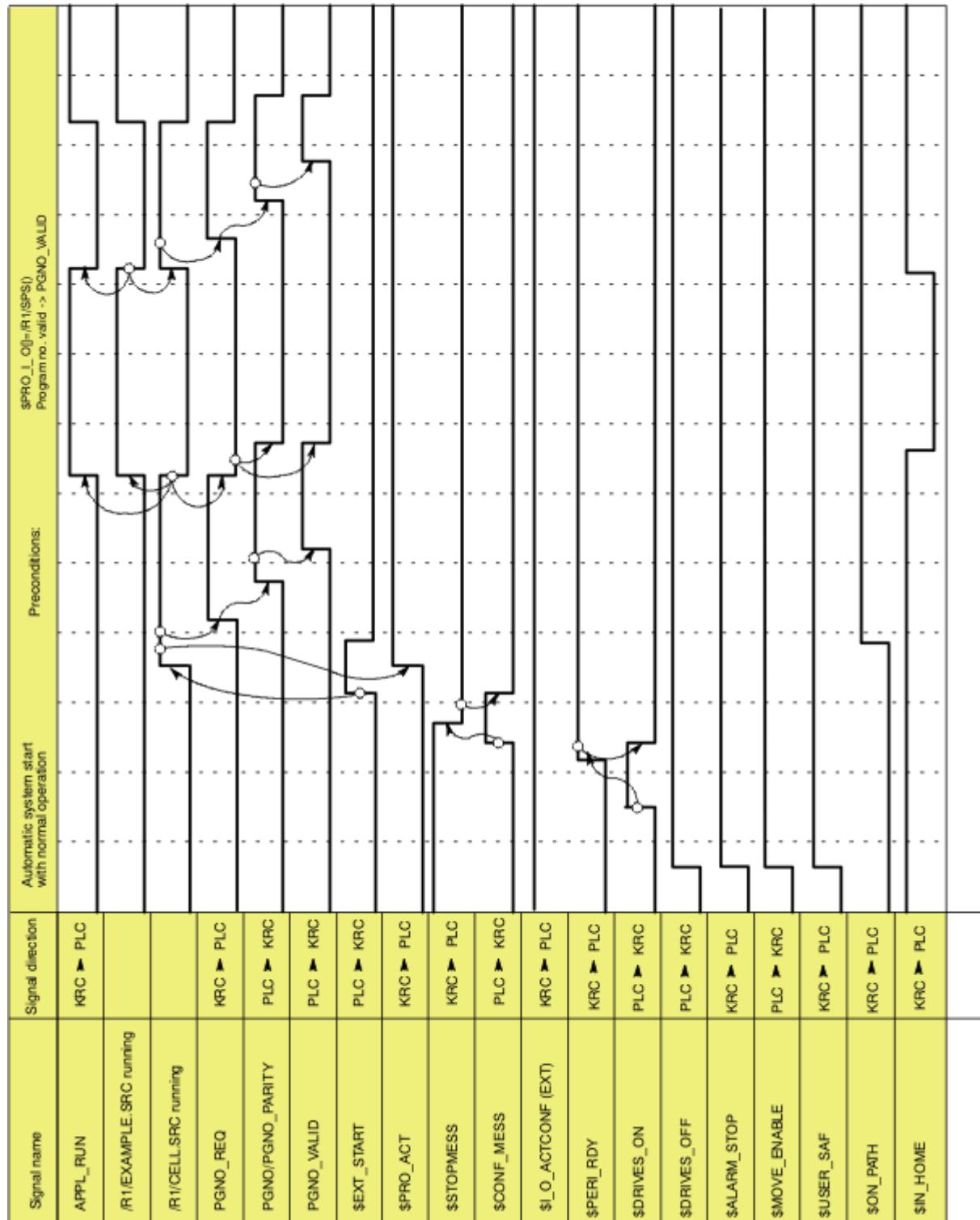
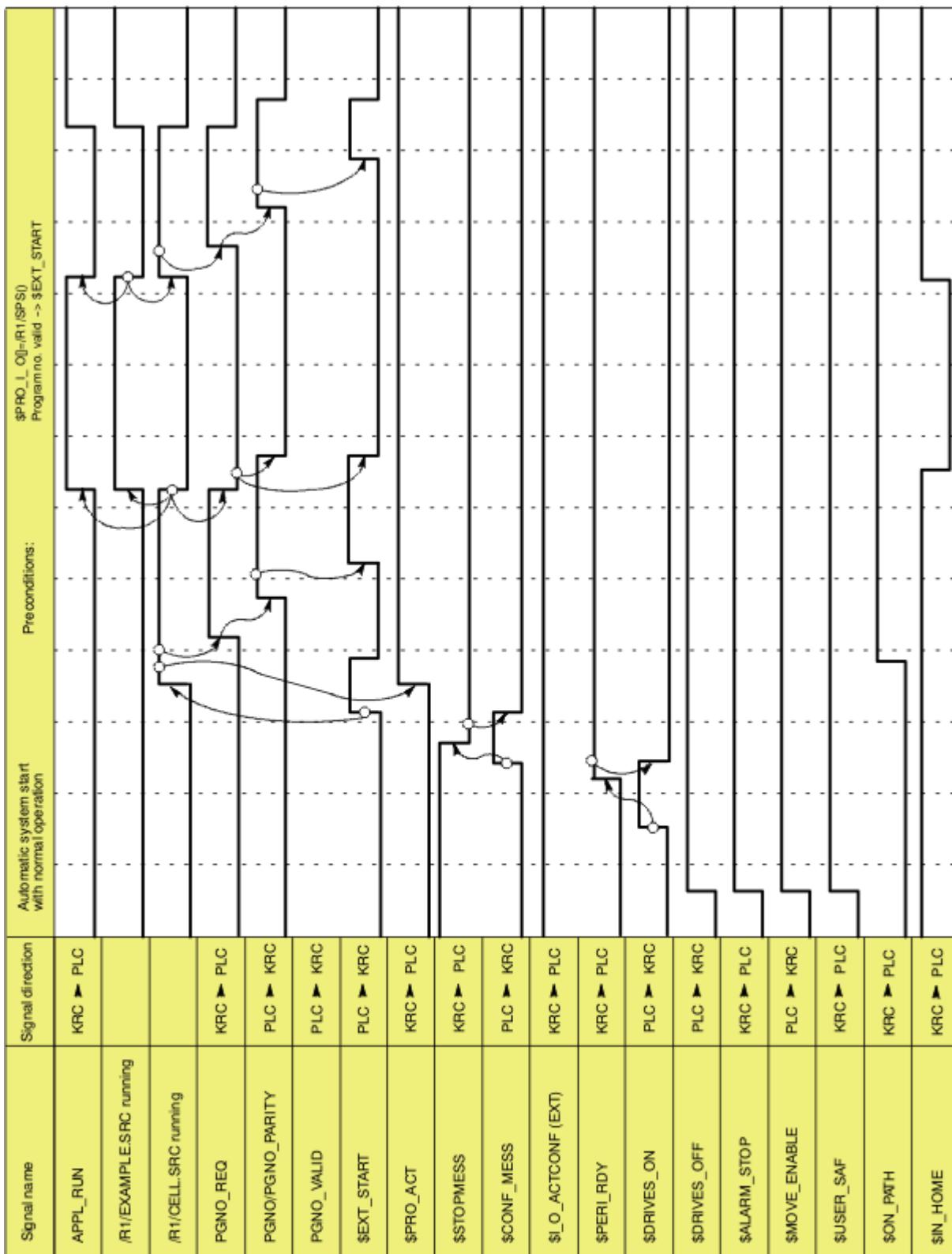
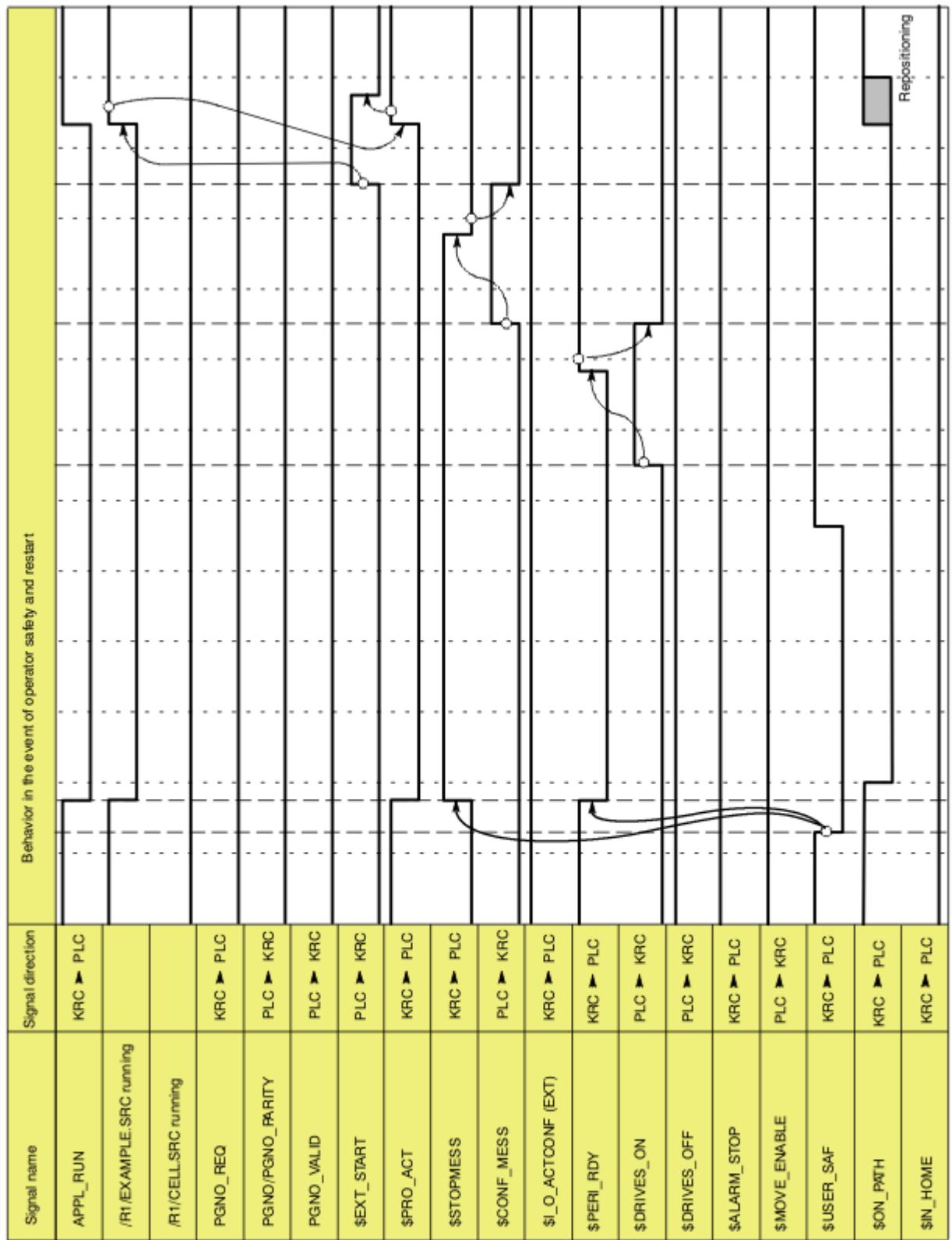


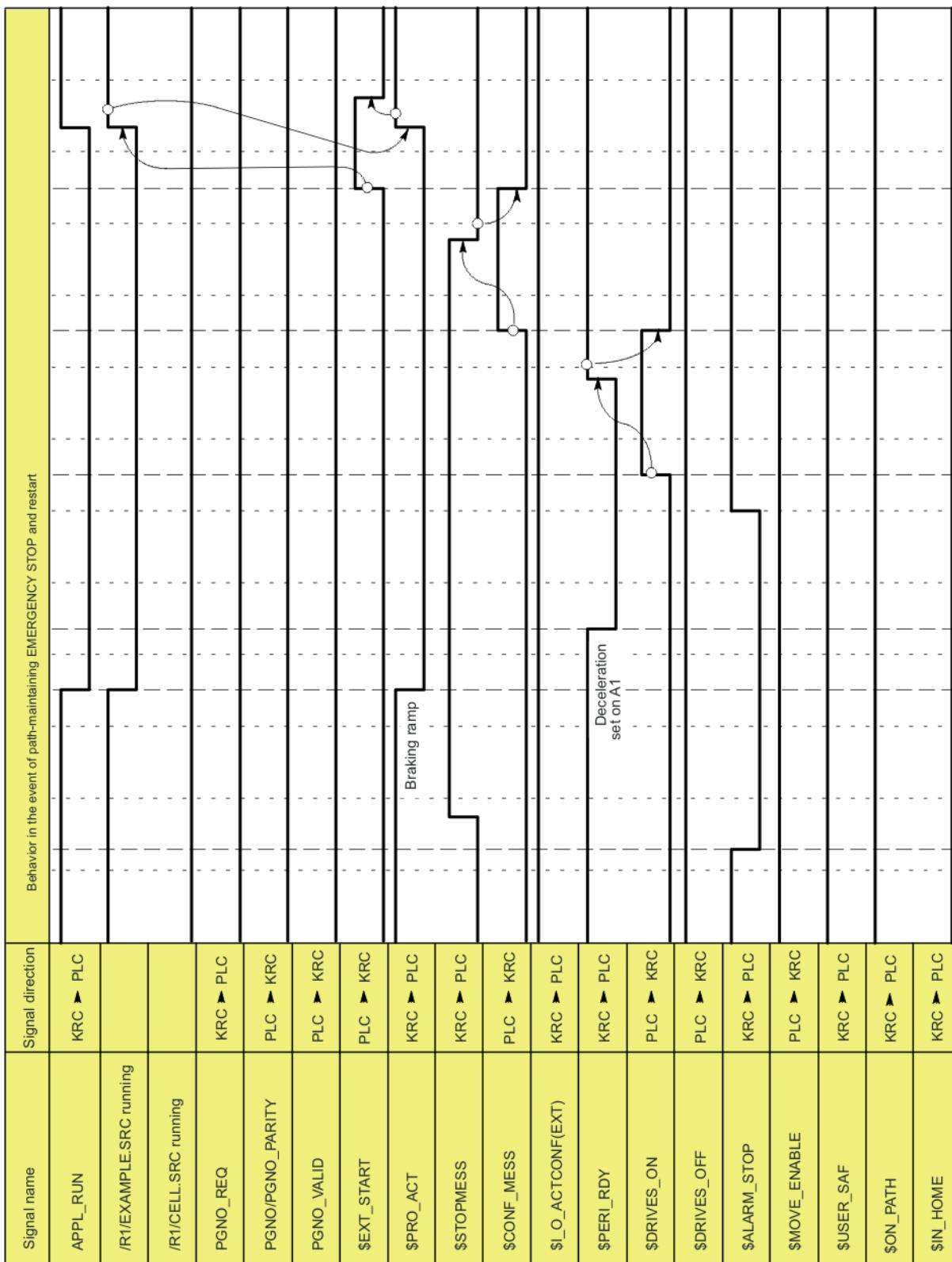
Fig. 6-24: Automatic system start and normal operation with program number acknowledgement by means of PGNO\_VALID



**Fig. 6-25: Automatic system start and normal operation with program number acknowledgement by means of \$EXT\_START**



**Fig. 6-26: Restart after dynamic braking (operator safety and restart)**



**Fig. 6-27: Restart after path-maintaining EMERGENCY STOP**

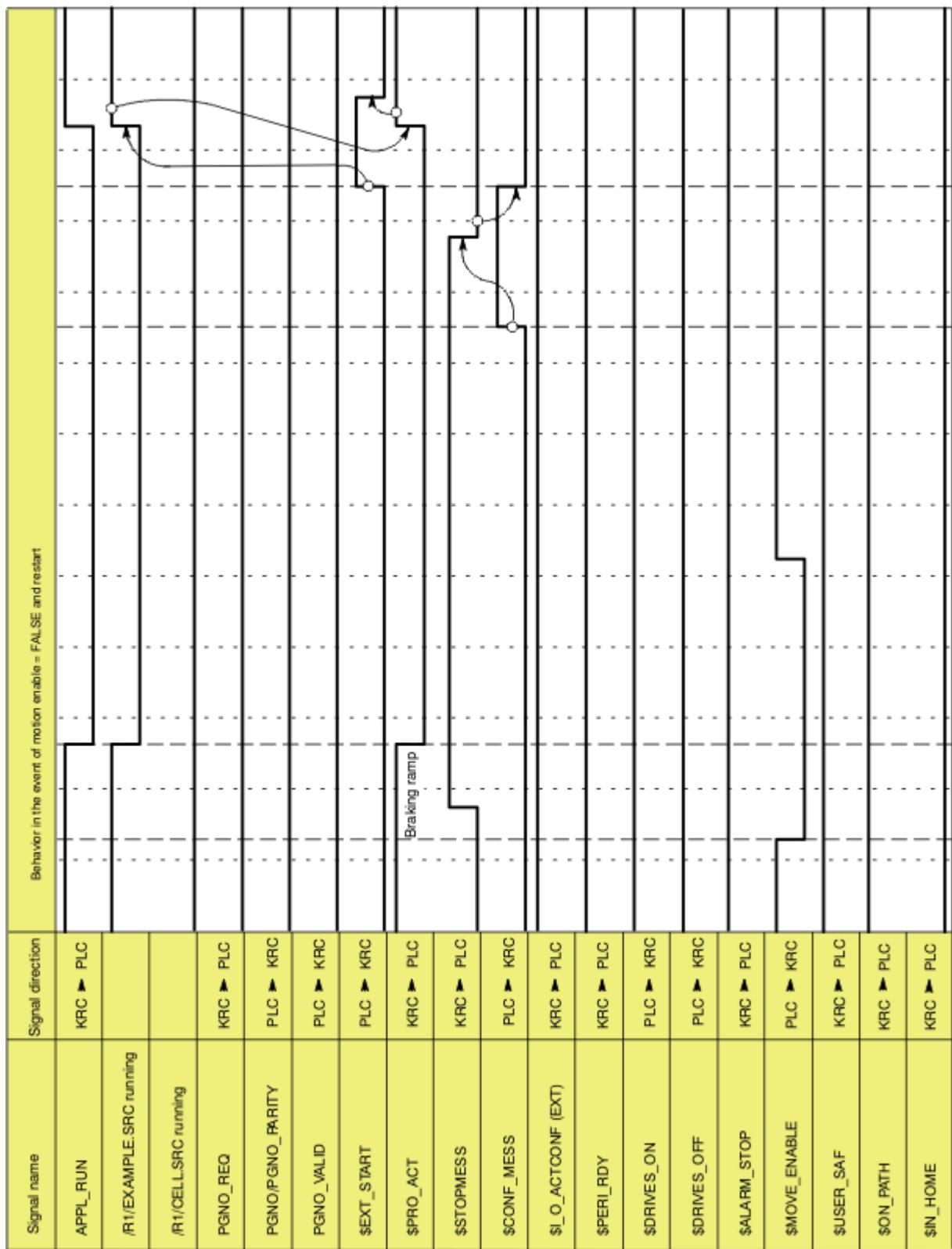


Fig. 6-28: Restart after motion enable

Signal name	Signal direction	Behavior in the event of user STOP and restart
APPL_RUN	KRC ➤ PLC	
/R1/EXAMPLE.SRC running		
PGNO_REQ	KRC ➤ PLC	
PGNO/PGNO_PARITY	PLC ➤ KRC	
PGNO_VALID	PLC ➤ KRC	
\$EXT_START	PLC ➤ KRC	
\$PRO_ACT	KRC ➤ PLC Programmed user STOP	
\$STOPMESS	KRC ➤ PLC	
\$CONF_MESS	PLC ➤ KRC	
\$I_O_ACTCONF (EXT)	KRC ➤ PLC	
\$PERI_RDY	KRC ➤ PLC	
\$DRIVES_ON	PLC ➤ KRC	
\$DRIVES_OFF	PLC ➤ KRC	
\$ALARM_STOP	KRC ➤ PLC	
\$MOVE_ENABLE	PLC ➤ KRC	
\$USER_SAF	KRC ➤ PLC	
\$ON_PATH	KRC ➤ PLC	
\$IN_HOME	KRC ➤ PLC	

Fig. 6-29: Restart after user STOP

## 6.20 Torque mode

### 6.20.1 Overview of torque mode

**Description** The “torque mode” function consists of the sub-functionalities “torque limitation” and “deactivation of monitoring functions”.

#### Torque limitation:

The torques, i.e. the motor current, can be limited for individual axes or multiple axes. Torque limitation enables the following applications:

- The axis can push or pull with a defined torque against a resistance.

Example:

Application of a defined pressure on the workpiece by an electric motor-driven spot welding gun.

- The axis can be set to “soft”. It can then be moved by application of an external force. It can be pushed away, for example.

Examples:

The robot must grip a workpiece in a press that is then ejected by the press. In order for the robot to be able to yield and absorb the ejector stroke, the affected axis is set to “soft”.

The robot must set a workpiece down at a point from which it can be pulled into exactly the right position by means of clamps. The robot must be compliant for this.

#### Deactivation of monitoring functions:

The torque limitation generally results in a relatively large deviation between the command position and the actual position. Certain monitoring functions are triggered by this deviation, although this is undesirable with torque limitation. These regular monitoring functions can thus be deactivated.

#### Restriction

The following restriction must be taken into consideration if axes are to absorb ejector motions:

A diagonal ejector motion cannot generally be absorbed by switching a single axis to “soft”. Remedy:

- In the case of slightly diagonal ejector motions, a possible remedy is to install the robot with a slight inclination.
- Or contact KUKA Deutschland GmbH.



Inclined installation of the robot is only permissible up to a certain angle of inclination. Further information is contained in the robot operating or assembly instructions.

#### 6.20.1.1 Using torque mode

Torque mode is only possible in program mode, not in manual mode.

**WARNING**

By default, the robot controller is configured so that only limits that exceed the holding torque of the axis (\$HOLDING\_TORQUE) can be set. It is nonetheless possible that the axis with limited torque is no longer able to achieve the necessary torque for braking, holding or moving the axis. This can be the case, for example, if the default configuration of the robot controller has been changed or incorrect load data are used.

Incorrectly set values can result in unexpected behavior of the robot controller, e.g. motion in a different direction or with different acceleration.

For this reason:

- Only ever limit the torque in small steps, gradually approaching the required limit.
- Do not limit the torque further than necessary.

Failure to take this precaution into consideration may result in death, injuries or damage to property.



If an application requires torque limits that no longer exceed the holding torque of the axis, KUKA Deutschland GmbH must be contacted.

**Procedure**

1. Set the torque limits for the desired axis and/or deactivate the regular monitoring functions.

(>>> 6.20.2 "Activating torque mode: SET\_TORQUE\_LIMITS()" Page 215)

If the regular monitoring functions are deactivated, other monitoring functions specially adapted to torque mode are automatically activated.

(>>> "Monitoring functions" Page 214)

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.
6. Deactivate torque mode again.

(>>> 6.20.3 "Deactivating torque mode: RESET\_TORQUE\_LIMITS()" Page 218)

The torque limits are canceled and the regular monitoring functions are reactivated. Furthermore, the command position is adjusted to the actual position.

**Activated/deactivated**

Torque mode is considered to be activated in the following case:

- If the upper torque limit is less than or equal to the upper value of the \$TORQUE\_AXIS\_MAX interval.  
(>>> 6.20.5.3 "\$TORQUE\_AXIS\_MAX" Page 220)
- 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:

- End of program
- 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: <i>Ackn. Max. following error exceeded ({Drive})</i> .
Standstill monitoring	1100: <i>Stopped {({Axis number})}</i>
Positioning monitoring	1105: <i>Positioning monitoring {({Axis number})}</i>
Monitoring whether motor blocked	26009: <i>Motor blocked ({Drive})</i> .

It is also possible, however, to retain the regular monitoring functions in torque mode. This may be useful, for example, if torque mode is used to avoid damage in the case of collisions.

(>>> 6.20.6.2 "Robot program: avoiding damage in the event of collisions"  
Page 223)

#### 6.20.1.2 Robot program example: setting A1 to "soft" in both directions

##### Description

This simple example illustrates the basic principle of torque mode.

In this example, A1 is to be set to "soft" in both directions. For this purpose, both the positive and negative current limits are set to 0 Nm. This allows A1 to be moved by application of an external force.

##### Program

```

...
1 PTP {A1 10}
2 SET_TORQUE_LIMITS(1, {lower 0, upper 0, monitor #off})
3 PTP {A1 11}
...
4 RESET_TORQUE_LIMITS(1)
5 PTP {A1 -20}
...
```

Line	Description
2	Negative and positive current limits of A1 are set to 0; the regular monitoring functions are deactivated.  (The actual velocity and the following error are now monitored with special monitoring functions.)
3	A "motion" is executed to activate torque limitation. (Since both current limits are set to 0, the robot will not actually move.)  A1 can now be moved by application of an external force.
4	Deactivate torque mode again for A1.  Torque limitation is canceled and the regular monitoring functions are reactivated. Furthermore, the command position is automatically adjusted to the actual position.
5	The robot moves to the next position.  (The motion from line 4 is not belatedly executed now, as the command/actual adjustment has been carried out in line 5.)

It is only for A1 that the holding torque is 0 Nm. The limits can therefore not generally be set to 0 for setting an axis to "soft".

This documentation contains a more detailed example of setting axes to "soft" that can also be applied to other axes.

(>>> 6.20.6.1 "Robot program: setting axis to "soft" in both directions"  
Page 222)

## 6.20.2 Activating torque mode: SET\_TORQUE\_LIMITS()

- |                    |                                                                                                                                                                                                                                                                                                                                                      |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | This function can be used to perform the following actions for a specific axis:                                                                                                                                                                                                                                                                      |
|                    | <ul style="list-style-type: none"> <li>■ Limit the torques in the positive and/or negative direction.</li> <li>■ Deactivate the regular monitoring functions that would be triggered by a higher following error.</li> <li>■ If the regular monitoring functions are deactivated: modify the values for the special monitoring functions.</li> </ul> |

**Function**      `SET_TORQUE_LIMITS (axis: in, values: in)`

Element	Description
<code>axis</code>	Type: INT  Axis to which the statement applies
<code>values</code>	Type: TorqLimitParam  Values set for the axis

### TorqLimitParam

STRUC TorqLimitParam REAL lower, upper, SW\_ONOFF monitor,  
REAL max\_vel, max\_lag

Element	Description
<code>lower</code>	Lower torque limit  Unit: Nm (for linear axes: N)  Default value: -1E10 (i.e. unlimited)
<code>upper</code>	Upper torque limit  Unit: Nm (for linear axes: N)  Default value: 1E10 (i.e. unlimited)

Element	Description
monitor	<ul style="list-style-type: none"> <li>■ <b>#ON</b> (default): Activates the regular monitoring functions.</li> <li>■ <b>#OFF</b>: Deactivates the regular monitoring functions. Instead, the monitoring functions <code>max_vel</code> and <code>max_lag</code> are activated.</li> </ul>
<code>max_vel</code>	<p>Maximum permissible actual velocity in torque mode (only relevant if the regular monitoring functions are deactivated)</p> <p>Only a positive value may be programmed.</p> <p>Unit: Degrees (for linear axes: mm)</p> <p>Default value (valid for all operating modes): T1 jog velocity * internal safety factor</p> <p>In T1, the maximum velocity with which jogging can be carried out is the default value, even if a higher value is programmed.</p> <p><b>Note:</b> Only set a higher value than the default value if absolutely necessary.</p>
<code>max_lag</code>	<p>Maximum permissible following error in torque mode (only relevant if the regular monitoring functions are deactivated)</p> <p>Only a positive value may be programmed.</p> <p>Unit: Degrees (for linear axes: mm)</p> <p>Default value: 5 degrees (for linear axes: 100 mm)</p> <p><b>Note:</b> Only set a higher value than the default value if absolutely necessary.</p>

**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 submit programs.
- **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  $\geq$  `lower`.
- If one limit (or both) is already set and the other limit is then set, and if the new limit would result in an empty interval, the new limit value becomes the value for both limits. Example:
  - Already set: {`lower 1, upper 2`}
  - Newly set: {`lower 3`}
  - This results in: {`lower 3, upper 3`}
- It is permissible to set a positive `lower` or a negative `upper` limit.
- The limits set must be greater than the current holding torque `$HOLDING_TORQUE`. If they are set differently, the robot controller generates an error message that must be acknowledged by the user.



If an application requires torque limits that no longer exceed the holding torque of the axis, KUKA Deutschland GmbH must be contacted.

- lower must be less than or equal to the upper value of the \$TORQUE\_AXIS\_MAX\_0 interval.
- upper must be greater than or equal to the lower value of the \$TORQUE\_AXIS\_MAX\_0 interval.
- If the limits are set differently, the robot controller generates an error message that must be acknowledged by the user.

## Examples

### Example 1:

For A1, the permissible torque range is limited to the interval 800 ... 1,400 Nm.

```
SET_TORQUE_LIMITS(1, {lower 800, upper 1400} )
```

### Example 2:

For A3, the upper torque limit is set to 1,200 Nm.

```
SET_TORQUE_LIMITS(3, {upper 1200} )
```

### Example 3:

For A1, the regular monitoring functions are (re)activated.

```
SET_TORQUE_LIMITS(1, {monitor #on} )
```

### Example 4:

For A1, the permissible torque range is limited to the interval -1,000 ... 1,000 Nm. Furthermore, the regular monitoring functions are deactivated and the special monitoring functions are set to user-defined values.

```
SET_TORQUE_LIMITS(1, {lower -1000, upper 1000, monitor #off, max_vel  
10, max_lag 20} )
```

### Example 5:

For A1, the permissible torque range is set to -1E10 ... 1E10, i.e. the range is unlimited. The regular monitoring functions are (re)activated.

```
SET_TORQUE_LIMITS(1, {lower -1E10, upper 1E10, monitor #on} )
```

This all corresponds to RESET\_TORQUE\_LIMITS (1), with the difference that in example 5, the command position is not to adapted to the actual position.

### Example 6:

For A1, the lower torque limit is set to a calculated value.

The value has been calculated with the function myCalc() and transferred with the variable myLimits. (In the concrete application, the user must write his own function for this.)

In order for the other components to be non-initialized, the value is pre-initialized with a partially initialized aggregate.

```
DECL TorqLimitParam myParams  
...  
myParams = {lower 0}  
myParams.lower = myCalc()  
SET_TORQUE_LIMITS(1, myParams)
```

### Example 7:

In this case, the limits are also set to a value that has been calculated with a function. (In the concrete application, the user must write his own function for this.)

The return value of the function is transferred directly, however.

```
DEFFCT TorqLimitParam myCalcLimits()
DECL TorqLimitParam myLimits
...
RETURN myLimits
ENDFCT
...
SET_TORQUE_LIMITS(1, myCalcLimits())
```

### 6.20.3 Deactivating torque mode: RESET\_TORQUE\_LIMITS()

#### Description

This function has the following effect on the selected axis:

- It cancels the limitation of the torques insofar as they were limited.
- It reactivates the regular monitoring functions insofar as they were deactivated.
- It adapts the command position to the actual position.

#### Function

`RESET_TORQUE_LIMITS (axis: in)`

Element	Description
<code>axis</code>	Type: INT Axis to which the statement applies

#### Properties

- The statement can be used in robot programs and in submit programs.
- **Advance run stop:** In the robot program, the statement triggers an advance run stop. This cannot be masked with CONTINUE!

#### Alternative

If no command/actual value adjustment is required, torque mode can also be deactivated with `SET_TORQUE_LIMITS` instead of `RESET_TORQUE_LIMITS`:

```
SET_TORQUE_LIMITS(1, {lower -1E10, upper 1E10, monitor #on} )
```

- Advantage: Can be used during a motion ("on the fly").
- Disadvantage: If the torque limitation has resulted in a relatively large following error, the robot accelerates very fast. This can trigger monitoring functions and stop the program.



Deactivation by means of `SET_TORQUE_LIMITS` is not suitable in most cases.

### 6.20.4 Interpreter specifics

#### Description

- `SET_TORQUE_LIMITS()` and `RESET_TORQUE_LIMITS()` can be used in robot programs and in submit programs.
- The statements are interpreter-specific, i.e. they only work in the interpreter in which they have been used.
- `SET_TORQUE_LIMITS()` first takes effect when the axis is moved for the interpreter that generates the statement. Example:
  - a. Torque mode is activated in the robot program for an external axis.
  - b. The external axis is moved by a submit program. Torque mode has no effect.
  - c. The external axis is moved by a robot program. Torque mode takes effect.

- If torque mode is already active, SET\_TORQUE\_LIMITS() takes effect immediately.
- SET\_TORQUE\_LIMITS() works immediately if a motion is active. For this reason, the torque limits can be set at any time in robot programs, both inside and outside an interrupt, and the monitoring functions can be activated and deactivated.

It is also possible to use torque mode inside an interrupt program only. (If RESET\_TORQUE\_LIMITS() is used, it may subsequently be necessary to return to the interrupt position with PTP \$AXIS\_RET.)

(>>> 6.20.6.3 "Robot program: torque mode in the interrupt" Page 224)

- When a torque-driven axis "changes owner", the command position is adjusted to the actual position.

"Change of owner" means: an interpreter has moved the axis in torque mode (and thus "owns" it). While torque mode is active, the axis is moved by a different interpreter.

The main application here is: jogging after a program has been interrupted in torque mode.

## Example

The following example shows when SET\_TORQUE\_LIMITS() is effective, depending on whether torque mode is already active or not.

Initial situation (default): the monitoring functions are activated.

```

1 SET_TORQUE_LIMITS(1, {monitor #off})
2 HALT
3 PTP_REL {A1 10}
4 HALT
5 SET_TORQUE_LIMITS(1, {monitor #on})
6 HALT
7 PTP_REL {A1 15}
```

Line	Description
1	The monitoring functions for A1 are deactivated.
2	Here the monitoring functions are still activated.
3	The axis is moved. From here on, the statement SET_TORQUE_LIMITS is effective.
4	The monitoring functions are deactivated.
5	The monitoring functions are activated.
6	Here the monitoring functions are already activated. Because torque limitation was already active, the statement took effect immediately and not just after the next motion of this axis.

## 6.20.5 Diagnostic variables for torque mode

All these variables and constants are write-protected.

Their value is not dependent on the interpreter.

### 6.20.5.1 \$TORQUE\_AXIS\_ACT

**Variable**            \$TORQUE\_AXIS\_ACT[*axis number*]  
 Data type: REAL

**Description**            Current motor torque for axis [*axis number*]  
 Unit: Nm (for linear axes: N)

The value is only relevant if the brakes are released. If the brakes are applied, it is virtually zero.

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

(>>> 6.20.5.6 "Comparison: \$TORQUE\_AXIS\_ACT and \$HOLDING\_TORQUE " Page 221)

#### \$BRAKE\_SIG

The state of the brakes can be displayed by means of the system variable \$BRAKE\_SIG. The value of \$BRAKE\_SIG is a bit array: bit 0 corresponds to A1, bit 6 corresponds to E1.

- Bit n = 0: Brake is closed.
- Bit n = 1: Brake is open.

#### 6.20.5.2 \$TORQUE\_AXIS\_MAX\_0

##### Constant

\$TORQUE\_AXIS\_MAX\_0[*axis number*]

Data type: REAL

##### Description

Maximum permanent motor torque for axis [*axis number*] at velocity 0

The value specifies an interval: from -*value* to +*value*.

Unit: Nm (for linear axes: N)

**Advance run stop:** does not trigger an advance run stop.



*lower* must be less than or equal to the upper value of the \$TORQUE\_AXIS\_MAX\_0 interval.

*upper* must be greater than or equal to the lower value of the \$TORQUE\_AXIS\_MAX\_0 interval.

If the limits are set differently, the robot controller generates an error message that must be acknowledged by the user.

#### 6.20.5.3 \$TORQUE\_AXIS\_MAX

##### Constant

\$TORQUE\_AXIS\_MAX[*axis number*]

Data type: REAL

##### Description

Absolute maximum motor torque for axis [*axis number*]

The value specifies an interval: from -*value* to +*value*.

Unit: Nm (for linear axes: N)

**Advance run stop:** does not trigger an advance run stop.

#### 6.20.5.4 \$TORQUE\_AXIS\_LIMITS

##### Variable

\$TORQUE\_AXIS\_LIMITS[*axis number*]

Data type: TorqLimitParam

##### Description

Currently active motor torque limitation for axis [*axis number*]

Unit: Nm (for linear axes: N)

The variable is primarily intended for diagnosis via the variable correction function or variable overview.

##### Characteristics:

- If there are currently no limits active, upper and lower remain non-initialized.
- The component monitor is always initialized unless the axis does not exist.  
This is relevant, for example, in the case of 4-axis and 5-axis robots: if the entire array is displayed, the non-existent axes can be easily identified.  
Non-existent external axes are simply not displayed when the entire array is displayed.
- Max\_vel and max\_lag are non-initialized if monitor = #ON, as the regular monitoring functions are active in this case.  
If monitor = #OFF, the values of max\_vel and max\_lag are displayed. The display is irrespective of whether they have been set explicitly in the current program or whether the default values are being used.



The fact that certain components remain non-initialized under certain conditions, simplifies diagnosis for the user.

If the variable \$TORQUE\_AXIS\_LIMITS[] is accessed via KRL, however, the robot controller may regard the access as "invalid". Recommendation: Check the state of the variable with VARSTATE() prior to access.

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

#### 6.20.5.5 \$HOLDING\_TORQUE

**Variable** \$HOLDING\_TORQUE[axis number]

Data type: REAL

**Description** Holding torque for the robot axis [axis number]

Unit: Nm

The holding torque refers to the current actual position of the axis and the current load.

(For external axes, the value 0 N is always returned.)

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

(>>> 6.20.5.6 "Comparison: \$TORQUE\_AXIS\_ACT and \$HOLDING\_TORQUE " Page 221)



If the upper and lower torque limits are set to \$HOLDING\_TORQUE[] for all axes, the robot must remain stationary when the brakes are released.

If this is not the case, i.e. if the robot drifts, the load is not correctly configured.

#### 6.20.5.6 Comparison: \$TORQUE\_AXIS\_ACT and \$HOLDING\_TORQUE

In the case of a robot that is stationary with the brakes released, \$TORQUE\_AXIS\_ACT is not equal to \$HOLDING\_TORQUE, although this might be assumed.

Characteristics of \$HOLDING\_TORQUE:

- A calculated value that does not take friction into consideration. Control effects have no influence on the value.
- Is only dependent on the current position of the axis. Thus remains unchanged if the position remains constant.

Characteristics of \$TORQUE\_AXIS\_ACT:

- Calculated from the actual currents. Friction and control effects thus affect the value.
- Can change if the position remains constant.

The discrepancy between \$HOLDING\_TORQUE and \$TORQUE\_AXIS\_ACT is less than or equal to the current friction if the robot remains stationary for at least 1 second. A precondition is that the load data are correct.

## 6.20.6 Other examples

### 6.20.6.1 Robot program: setting axis to “soft” in both directions

#### Description

The robot must grip a workpiece in a press that is then ejected by the press. In order for the robot to be able to yield and absorb the ejector stroke, the axis is set to “soft”.

For this, the torque limits must be set to a very small interval around the holding torque. (It is only for A1 that the holding torque = 0 Nm. The limits can therefore not generally be set to 0 for setting an axis to “soft”.)

Assumption for this example: a rotation about the axis [*ideal\_axis*] moves the gripper almost exactly in the ejector direction.

#### Program

```

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 [ <i>ideal_axis</i> ], 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 [ <i>ideal_axis</i> ] can now be moved by the ejector of the press.
8	Signal to the press controller that the axis is ready for the ejection.
9	Wait for signal from press controller that the workpiece has been ejected.
10	Cancels torque limitation and reactivates the regular monitoring functions. Furthermore, adjusts the command position to the actual position.
11	Signal to the press controller that the axis is no longer ready for an ejection.
12	If required: Wait for a signal that motion away from the position is allowed.

### 6.20.6.2 Robot program: avoiding damage in the event of collisions

- Description** Torque limitation can be used to avoid damage in the event of collisions.
- Advantage: It is assured that the robot only presses against the obstacle with a defined, limited force.
  - Disadvantage: The robot becomes sluggish. High accelerations are no longer possible.

- Program** The robot fetches workpieces from a box. During the motion to points P7, P8 and P9, the possibility cannot be ruled out of the robot with the workpiece colliding with the box. It must be ensured that the robot does not press so hard that damage can result. For this, the forces are limited before the critical points.

The regular monitoring functions are deactivated. This is not because they would otherwise be triggered unnecessarily; on the contrary, they are not strict enough for this example. Instead, one of the special monitoring functions is set to a very low value. (Depending on the specific application, it may also be useful to use the regular monitoring functions.)

```

...
1 DECL TorqLimitParam myParams
...
2 FOR i = 1 to 6
3   myParams.lower = $holding_torque[i] - 500
4   myParams.upper = $holding_torque[i] + 500
5   myParams.monitor = #off
6   myParams.max_lag = 0.1
7   SET_TORQUE_LIMITS(i, myParams)
8 ENDFOR
9 $acc.cp = my_low_acceleration
10 $vel.cp = my_low_velocity
11 LIN P7
12 LIN P8
13 LIN P9
14 FOR i = 1 to 6
15   myParams.lower = -1E10
16   myParams.upper = 1E10
17   myParams.monitor = #on
18   SET_TORQUE_LIMITS(i, myParams)
19 ENDFOR
20 $acc.cp = my_high_acceleration
21 $vel.cp = my_high_velocity
22 LIN P10
...

```

Line	Description
2 ... 7	The torques for A1 ... A6 are limited.
3, 4	The limits are set to a relatively small interval centered on the holding torque.
5, 6	The regular monitoring functions are deactivated. Max_lag = 0.1 has the effect of triggering a stop in the case of a following error as low as 0.1°.
9, 10	Acceleration and velocity are reduced so that the robot approaches the critical point slowly.
11 ... 13	Points at which a collision could occur. If a collision occurs, the monitoring function max_lag is triggered and the system operator can intervene.
After the critical section:	

Line	Description
14 ...19	Torque limitation is deactivated. SET_TORQUE_LIMITS can be used here: the robot only reaches this point if it has passed the critical points without collision. In this case, no following error has built up and command/actual value adjustment is not required.
20, 21	Acceleration and velocity are reset to the previous higher values.
22	Uncritical point

### 6.20.6.3 Robot program: torque mode in the interrupt

- Description** This demonstrates that torque mode can be used completely inside an interrupt program. The example is not primarily a practical one, but is intended to show that this use is generally possible.
- Program** The robot is to determine the position of a workpiece whose possible location is known, but not its exact position. First, a proximity sensor signals to the robot controller when the robot is in the vicinity of the workpiece. On the basis of this sensor signal, an interrupt program is then called.
- The actual search is carried out in the interrupt program: A1 moves towards the workpiece. Torque mode is activated for A1 beforehand. The robot comes to a standstill where it makes contact with the workpiece due to the reduced torques: the workpiece has been “found”. This position can now be saved as the position of the workpiece.

```

...
1 LIN P1
2 INTERRUPT DECL 1 WHEN sensor_signal==true DO search()
3 INTERRUPT ON 1
4 LIN P2
5 INTERRUPT OFF 1
...
-----
6 DEF search()
7 ...
8 BRAKE
9 SET_TORQUE_LIMITS(1,{lower 1000, upper 1000, monitor #off})
10 PTP_REL {A1 10}
11 RESET_TORQ_LIMITS(1)
12 piece_found = $POS_ACT_MES
13 PTP $AXIS_RET
14 END

```

Line	Description
1 ... 4	On the way to P2, the sensor is to signal when the robot is in the vicinity of the workpiece.
2	When the sensor signal is received, the subprogram search() is called.
<b>In the subprogram:</b>	
8	The current motion is stopped as soon as search() is executed.
9	Sets torque limits for A1, deactivates regular monitoring functions.

Line	Description
10	A1 moves. The robot comes to a standstill where it makes contact with the workpiece due to the reduced torques.  Once the robot has reached its command position {A1 10}, the next program line is executed. (The fact that the actual position deviates from the command position has no effect, as the regular monitoring functions have been deactivated.)
11	Cancels torque limitation and reactivates the regular monitoring functions. Furthermore, adjusts the command position to the actual position.
12	The current position of the robot now indicates the position of the workpiece. This is saved here in a variable.
13	Returns to the position at which the robot left the path in the main program.

#### 6.20.6.4 Robot program: servo gun builds up pressure

**Description** This program shows how torque mode can be activated by means of a trigger. (Comparable programs are used in the background in the KUKA.ServoGun technology packages. They thus do not need to be programmed by the user.)

**Program** **Main program:**

```

1 DEF SPOT()
2 DECL BOOL error_occurred
...
3 Interrupt DECL 1 WHEN $stopmess DO resume_subprog()
4 Interrupt ON 1
5 REPEAT
6   error_occurred = false
7   SPOT_MOVE()
8 UNTIL error_occurred == false
...

```

Line	Description
3	If an error occurs, resume_subprog() is to be called.
7	The weld program SPOT_MOVE() is called.
5 ... 8	If an error has occurred (i.e. if error_occurred == true), SPOT_MOVE() is repeated.

**Weld program:**

```

1 DEF SPOT_MOVE()
...
2 TorqLimWeld = {lower -1000, upper 1000 , monitor #off}
3 i = 6+EG_EXTAX_ACTIVE
...
4 LIN P_APPROX C_DIS
5 $VEL_EXTAX[EG_EXTAX_ACTIVE]=EG_MAX_CONST_VEL[EG_EXTAX_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 the actual position.

#### Interrupt program in the event of an error:

```
1 DEF resume_subprog()
2 BRAKE
3 Suppress_repositioning()
4 HALT
5 error_occurred = true
6 RESUME
7 END
```

Line	Description
3	Normally, in the case of a restart after HALT, the robot is repositioned to the position at which the interrupt was triggered. (This is because of \$STOPMESS.) Suppress_repositioning() prevents this repositioning. Suppress_repositioning() may be useful, depending on the application, but not necessarily.
5	Set error_occurred to TRUE so that the REPEAT loop in the main program is repeated.
6	RESUME deactivates torque mode. Jump back to line 8 in the main program.

#### 6.20.6.5 Submit program: servo gun builds up pressure

##### Precondition

- E1 is already asynchronous with ASYPTP {E1 10}.
- or: \$ASYNC\_MODE is configured in such a way (bit 0 = 1) that the axis is implicitly set to synchronous mode in the case of ASYPTP in the submit program.

##### Program

```
...
1 IF $PRO_STATE1==#P_FREE
2   SET_TORQUE_LIMITS(7,{upper 1000, monitor #off })
3   ASYPTP {E1 10}
...
4   RESET_TORQUE_LIMITS(7)
5   ASYPTP {E1 -10}
6 ENDIF
...
```

Line	Description
1	Ensure that no robot program is selected.
2	Limit the positive torque and deactivate the regular monitoring functions.
3	Motion towards end point {E1 10} behind the workpiece. The pressure on the workpiece builds up.

Line	Description
4	Cancels torque limitation and reactivates the regular monitoring functions. Furthermore, adjusts the command position to the actual position.  The interpreter waits in RESET_TORQUE_LIMITS(7) until the asynchronous motion has been completed. Only then does it perform the command/actual value adjustment. It is therefore not necessary to program WAIT FOR \$ASYNC_STATE == #IDLE before "RESET...".
5	Reopen the gun.

## 6.21 Event planner: Configuring a data comparison

**Description** This function can be used for time-related or action-related control of the data comparison between the kernel system and the hard drive. During data comparison, the kernel system data are written to the hard drive.

**Precondition**

- “Expert” user group

**Procedure**

1. In the main menu, select **Configuration > Miscellaneous > Event planner**.
2. Expand the tree structure in the left-hand section of the window and select the desired action.

### Scheduled actions

- **T1 and T2 consistency:** Compare data in operating modes T1 and T2 at regular intervals.
- **AUT and EXT consistency:** Compare data in the automatic operating modes at regular intervals.

### Conditional actions

- **Logic consistency:** Compare data after a change of operating mode or after online optimizing.  
  
Online optimizing is the modification of the parameters of a program during operation.
- 3. Make the desired settings in the right-hand part of the window. Depending on the selected action, various parameters are available for selection, e.g. various time intervals.
- 4. Press **Save**.

## 6.22 Brake test

### 6.22.1 Overview of the brake test

**Description** Each robot axis has a holding brake integrated into the motor. The brake test checks every axis at low speed and at the current temperature to see if the braking torque is sufficiently high, i.e. whether it exceeds a certain minimum value. The minimum value for the individual axes is stored in the machine data. (The brake test does not calculate the absolute value of the braking torque.)

**Request** If the brake test is active, the following events cause a brake test to be requested:

- Input \$BRAKETEST\_REQ\_EX is set externally, e.g. by a PLC (external request)
- Robot controller boots with a cold start (internal request)
- Function test of the brake test (internal request)

- Brake test cycle time has elapsed (internal request)

#### Cycle time

The cycle time is 46 h. It is deemed to have elapsed when the drives have been under servo-control for a total of 46 h. The robot controller then requests a brake test and generates the following message: *Brake test required*. The robot can be moved for another 2 hours. It then 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.

#### Execution

A precondition for the brake test is that the robot is at operating temperature. This is the case after approx. 1 h in normal operation.

The brake test is carried out using the program BrakeTestReq.SRC. It can be started in the following ways:

- **Automatically**

Integrate BrakeTestReq.SRC into the application program in such a way that it is cyclically called as a subprogram. If a brake test is requested, the robot detects this and starts the brake test.

- **Manually**

Start the program BrakeTestReq.SRC manually.

#### Sequence

The brake test checks all brakes one after the other.

1. The robot accelerates to a defined velocity. (The velocity cannot be influenced by the user.)
2. Once the robot has reached the velocity, the brake is applied and the result for this braking operation is displayed in the message window.
3. If a brake has been identified as being defective, the brake test can be repeated for confirmation or the robot can be moved to the parking position. 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 a brake has been identified as being defective, the drives remain under servo-control for 2 hours following the start of the brake test (= monitoring time). The robot controller then switches the drives off.

#### Overview

Step	Description
<b>In WorkVisual:</b>	
1	If required: Activate the brake test in WorkVisual.  (>>> 6.22.2 "Activating the brake test" Page 229)
<b>On the robot controller:</b>	
2	Configure input and output signals for the brake test.  (>>> 6.22.4 "Configuring input and output signals for the brake test" Page 230)
3	Teach positions for the brake test.  The parking position must be taught. The start position and end position can be taught.  (>>> 6.22.5 "Teaching positions for the brake test" Page 233)
4	If the brake test is to be carried out automatically:  Integrate BrakeTestReq.SRC into the application program in such a way that it is cyclically called as a subprogram.

Step	Description
5	If the brake test is to be carried out manually: Start the program BrakeTestReq.SRC manually. (>>> 6.22.6 "Performing a manual brake test" Page 233)
6	If required: Test the function of the brake test. (>>> 6.22.7 "Checking that the brake test is functioning correctly" Page 234)

### 6.22.2 Activating the brake test

- If a safety option is installed and the safe monitoring is active, the brake test is automatically active.
- If the brake test is not automatically active, the user has the option of manually activating it. This must be carried out in WorkVisual.



If the brake test is not automatically active, the user must carry out a risk assessment to determine whether it is necessary to activate the brake test for the specific application.



Further information about activating the brake test is contained in the WorkVisual documentation.

### 6.22.3 Programs for the brake test

The programs are located in the directory C:\KRC\ROBOT-ER\KRC\R1\TP\BrakeTest.

Program	Description
BrakeTestReq.SRC	<p>This program performs the brake test.</p> <p>It can be performed in the following ways:</p> <ul style="list-style-type: none"> <li>■ Integrate the program into the application program in such a way that it is cyclically called as a subprogram. If a brake test is requested, the robot detects this and performs the brake test immediately.</li> <li>■ Execute the program manually.</li> <li>■ Test the function of the brake test. The robot controller executes BrakeTestReq.SRC with special parameterization.</li> </ul>
BrakeTestPark.SRC	<p>The parking position of the robot must be taught in this program.</p> <p>The robot can be moved to the parking position if a brake has been identified as being defective. Alternatively, the brake test can be repeated for confirmation.</p>
BrakeTestStart.SRC	<p>The start position of the brake test can be taught in this program. The robot starts the brake test from this position.</p> <p>If the start position is not taught, the robot performs the brake test at the actual position.</p>
BrakeTestBack.SRC	<p>The end position of the brake test can be taught in this program. The robot moves to this position after the brake test.</p> <p>If the end position is not taught, the robot remains at the actual position after the brake test.</p>
BrakeTestSelf-test.SRC	<p>The program checks whether the brake test has correctly detected a defective brake. For this purpose, the robot controller executes BrakeTestReq.SRC with special parameterization.</p>

#### 6.22.4 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:\STEUMADA.



**WARNING** These signals are not redundant in design and can supply incorrect information. Do not use these signals for safety-relevant applications.

**Precondition**

- “Expert” user group

**Procedure**

1. Open the file \$machine.dat in the directory KRC:\STEUMADA 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 compelling need to assign output numbers to them. It is only necessary to assign numbers if there is a need to be able to read the signals (e.g. via the variable correction function or program execution.)

Signal	Description
\$BRAKETEST_REQ_EX	Input <ul style="list-style-type: none"> <li>■ <b>TRUE</b> = 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.</li> <li>■ <b>FALSE</b> = brake test is not being requested externally.</li> </ul>
\$BRAKETEST_MONTIME	Output <ul style="list-style-type: none"> <li>■ <b>TRUE</b> = robot was stopped due to elapsed monitoring time. Acknowledgement message 27002 is generated.</li> <li>■ <b>FALSE</b> = acknowledgement message 27002 is not active. (Not generated, or has been acknowledged.)</li> </ul>
\$BRAKETEST_REQ_INT	Output <ul style="list-style-type: none"> <li>■ <b>TRUE</b> = 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.</li> <li>■ <b>FALSE</b> = brake test is not requested (either internally or externally).</li> </ul>

Signal	Description
\$BRAKETEST_WORK	<p>Output</p> <ul style="list-style-type: none"> <li>■ <b>TRUE</b> = brake test is currently being performed.</li> <li>■ <b>FALSE</b> = brake test is not being performed.</li> </ul> <p>If no defective brakes have been detected, message 27012 is generated.</p> <p>Edge <b>TRUE → FALSE</b>:</p> <ul style="list-style-type: none"> <li>■ Test was successfully completed. No brake is defective. Message 27012 is generated.</li> <li>■ Or at least 1 defective brake was detected and the robot has moved to the parking position.</li> <li>■ Or the program was canceled during execution of the brake test.</li> </ul>
\$BRAKES_OK	<p>Output</p> <ul style="list-style-type: none"> <li>■ Edge <b>FALSE → TRUE</b>: Output was set to FALSE by the previous brake test. The brake test was carried out again and no defective brake was detected.</li> <li>■ Edge <b>TRUE → FALSE</b>: A brake has just been detected as defective. Message 27007 is generated.</li> </ul>
\$BRAKETEST_WARN	<p>Output</p> <ul style="list-style-type: none"> <li>■ Edge <b>FALSE → TRUE</b>: At least 1 brake has been detected as having reached the wear limit. Message 27001 is generated at the same time.</li> <li>■ Edge <b>TRUE → FALSE</b>: Output was set to TRUE by the previous brake test. The brake test was carried out again and no worn brake was detected.</li> </ul>

#### Messages

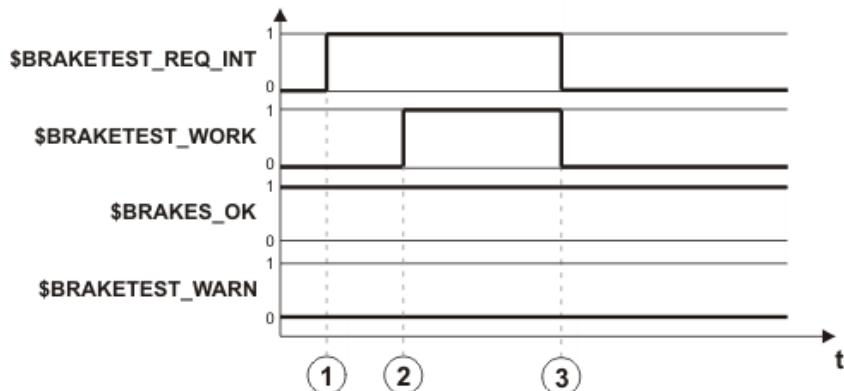
No.	Message
27001	<i>Brake {Brake no.}{Axis no.} has reached the wear limit</i>
27002	<i>Cyclical check for brake test request not made</i>
27004	<i>Brake test required</i>
27007	<i>Insufficient holding torque of brake {Brake no.}{Axis no.}</i>
27012	<i>Brake test successful</i>

#### 6.22.4.1 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.



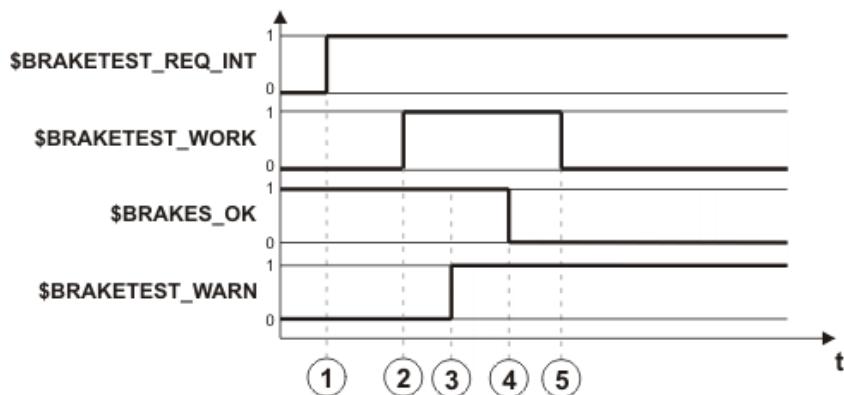
**Fig. 6-30: Signal diagram: brakes OK**

Item	Description
1	The brake test is requested.
2	Automatic call of the program BrakeTestReq.scc 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. 6-31: 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.scc 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.

## 6.22.5 Teaching positions for the brake test

<b>Description</b>	The parking position must be taught. The start position and end position can be taught. <ul style="list-style-type: none"> <li>■ If the start position is not taught, the robot performs the brake test at the actual position.</li> <li>■ If the end position is not taught, the robot remains at the actual position after the brake test.</li> </ul>
<b>Parking position</b>	If a brake is identified as being defective, the robot can be moved to the parking position. Alternatively, the brake test can be repeated for confirmation. <div style="border: 1px solid black; padding: 10px; margin-top: 10px;"> <p><b>⚠ WARNING</b> The parking position must be selected in a position where no persons are endangered if the robot sags because of the defective brake. The transport position, for example, can be selected as the parking position. Further information about the transport position is contained in the robot operating or assembly instructions.</p> </div>
<b>Precondition</b>	<ul style="list-style-type: none"> <li>■ All output signals are assigned to outputs.</li> <li>■ User group “Expert”</li> <li>■ Operating mode T1</li> </ul>
<b>Procedure</b>	<ol style="list-style-type: none"> <li>1. Open the program BrakeTestStart.SRC in the directory R1\TP\BrakeTest.</li> <li>2. Teach the motions to the start position of the brake test.           <ul style="list-style-type: none"> <li>■ The motions must be taught in such a way that the robot cannot cause a collision on the way to the start position.</li> <li>■ In the start position, every robot axis must have an available motion range of <math>\pm 10^\circ</math>.</li> </ul> </li> <li>3. Save and close the program.</li> <li>4. Open the program BrakeTestBack.SRC in the directory R1\TP\BrakeTest.</li> <li>5. Teach the motions from the start position to the end position of the brake test.           <p>The start and end position may be identical.</p> </li> <li>6. Save and close the program.</li> <li>7. Open the program BrakeTestPark.SRC in the directory R1\TP\BrakeTest.</li> <li>8. Program the motions from the start position to the parking position of the robot.</li> <li>9. Save and close the program.</li> </ol>

## 6.22.6 Performing a manual brake test

**⚠ WARNING** If a brake is identified as being defective and the drives are deactivated, the robot may sag. For this reason, no stop may be triggered during the motion to the parking position. The monitoring functions that can trigger a stop in this range (e.g. monitoring spaces) must be deactivated beforehand. No safety functions may be executed that would trigger a stop (e.g. E-STOP, opening the safety gate, change of operating mode, etc.).  
If a brake has been identified as being defective, the parking position must be approached no faster than at 10% of maximum velocity.



**WARNING** Program override for the test is automatically set to 100%. The robot moves at high velocity. Make sure that the robot cannot collide and that no persons are in the motion range of the robot.

**Precondition**

- No persons or objects are present within the motion range of the robot.
- In the start position, every robot axis has an available motion range of  $\pm 10^\circ$ . (Or, if no start position has been taught, in the actual position.)
- The parking position has been taught in the program BrakeTestPark.SRC.
- “Expert” user group
- Program run mode GO
- AUT mode
- The robot is at operating temperature (= after approx. 1 h in normal operation).

**Procedure**

1. Select the program BrakeTestReq.SRC in the directory R1\TP\BrakeTest and press the Start key.
2. The following message is displayed: **Performing manual brake test - please acknowledge**. Acknowledge the message.
3. Press the Start key. The message *Programmed path reached (BCO)* is displayed.
4. Press the Start key. The brakes are tested, starting with A1.
5. Possible results:
  - If a brake is OK, this is indicated by the following message: *Brake {Brake no.}{Axis no.} OK*.  
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.)  
Deselect the program BrakeTestReq.SRC.
  - If a brake is defective, this is indicated by the following message: *In-sufficient holding torque of brake {Brake no.}{Axis no.}*.  
Once all brakes have been tested, either press **Repeat** to repeat the brake test for checking purposes  
or press **Park pos.** to move the robot to the parking position.



If a brake has been identified as being defective, the drives remain under servo-control for 2 hours following the start of the brake test (= monitoring time). The robot controller then switches the drives off.

### 6.22.7 Checking that the brake test is functioning correctly

**Description**

It is possible to check whether the brake test has correctly detected a defective brake: the program BrakeTestSelfTest.SRC simulates a fault in the brakes and triggers a brake test. If the brake test detects the simulated fault, it is functioning correctly.



**WARNING** Program override for the test is automatically set to 100%. The robot moves at high velocity. Make sure that the robot cannot collide and that no persons are in the motion range of the robot.

**Precondition**

- No persons or objects are present within the motion range of the robot.
- In the start position, every robot axis has an available motion range of  $\pm 10^\circ$ . (Or, if no start position has been taught, in the actual position.)

- The parking position has been taught in the program BrakeTestPark.SRC.
- “Expert” user group
- Program run mode GO
- AUT mode
- The robot is at operating temperature (= after approx. 1 h in normal operation).

**Procedure**

1. Select the program BrakeTestSelfTest.SRC in the directory R1\TP\BrakeTest and press the Start key.
2. The following message is displayed: *Performing self-test for brake test - please acknowledge*. Confirm the message by pressing **Ackn..**.
3. Press the Start key.
4. Result of the function test:
  - Message *Insufficient holding torque of brake 3*: The brake test has correctly detected the simulated fault. The brake test is functioning correctly. Deselect the program BrakeTestSelfTest.SRC.  
Perform a manual brake test. This ensures that the simulated fault does not remain active.
  - Any other message, or no message, means: The brake test has not detected the simulated fault. The brake test is not functioning correctly.



**WARNING** If the function test establishes that the brake test is not functioning correctly:

- The robot must no longer be moved.
- KUKA Deutschland GmbH must be contacted.



## 7 Program and project management

### 7.1 Creating a new program

**Description** It is not possible to select a template in the user group “User”. By default, a program of type “Module” is created.

**Precondition**

- The Navigator is displayed.

**Procedure**

1. In the directory structure, select the folder in which the program is to be created, e.g. the folder **Program**. (Not all folders allow the creation of programs within them.)
2. Press **New**.
3. Only in the user group “Expert”:  
The **Template selection** window is opened. Select the desired template and confirm with **OK**.
4. Enter a name for the program and confirm it with **OK**.

### 7.2 Creating a new folder

**Precondition**

- The Navigator is displayed.

**Procedure**

1. In the directory structure, select the folder in which the new folder is to be created, e.g. the folder **R1**.  
Not all folders allow the creation of new folders within them. In the user groups “Operator” and “User”, new folders can only be created in the folder **R1**.
2. Press **New**.
3. Enter a name for the folder and confirm it with **OK**.

### 7.3 Renaming a file or folder

**Precondition**

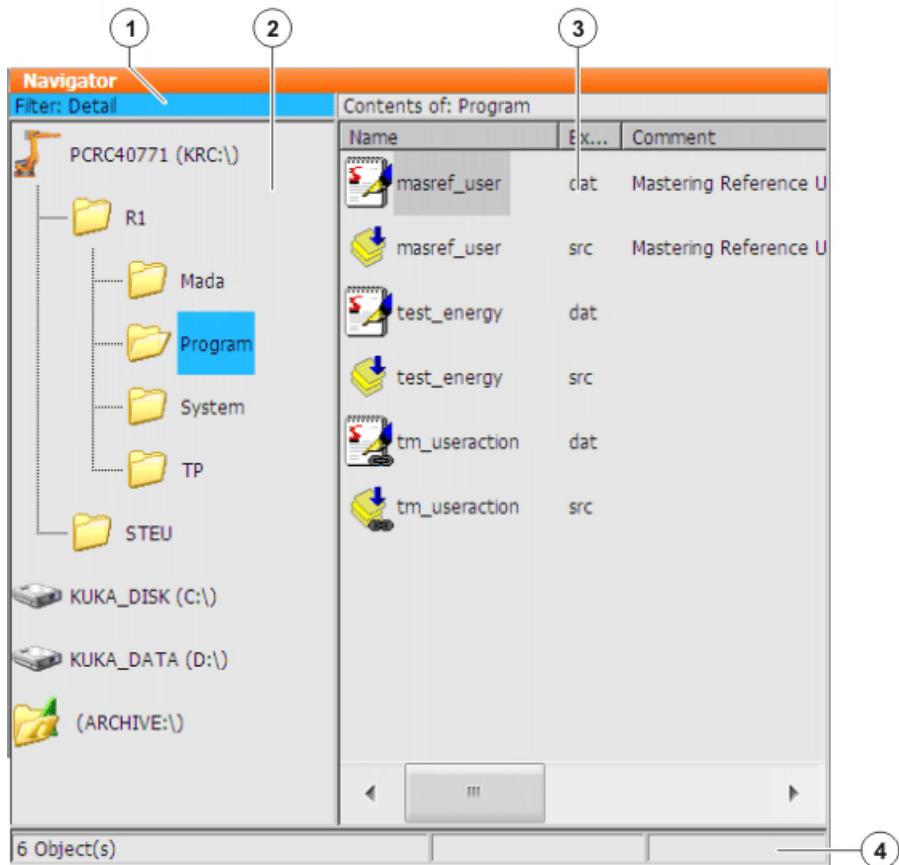
- The Navigator is displayed.

**Procedure**

1. In the directory structure, select the folder in which the file or folder to be renamed is located.
2. Select the file or folder in the file list.
3. Select **Edit > Rename**.
4. Overwrite the name with the new name and confirm with **OK**.

## 7.4 Navigator file manager

### Overview



**Fig. 7-1: Navigator**

- |                       |              |
|-----------------------|--------------|
| 1 Header              | 3 File list  |
| 2 Directory structure | 4 Status bar |

### Description

In the Navigator, the user manages programs and system-specific files.

#### Header

- Left-hand area: the selected filter is displayed.  
(>>> 7.4.1 "Selecting filters" Page 239)
- Right-hand area: the directory or drive selected in the directory structure is displayed.

#### Directory structure

Overview of directories and drives. Exactly which directories and drives are displayed depends on the user group and configuration.

#### File list

The contents of the directory or drive selected in the directory structure are displayed. The manner in which programs are displayed depends on the selected filter.

The file list has the following columns:

Column	Description
Name	Directory or file name
Extension	File extension This column is not displayed in the user group "User".

<b>Column</b>	<b>Description</b>
Comment	Comment
Attribute	Attributes of the operating system and kernel system This column is not displayed in the user group “User”.
Size	File size in kilobytes This column is not displayed in the user group “User”.
#	Number of changes made to the file
Changed	Date and time of the last modification
Created	Date and time of file creation This column is not displayed in the user group “User”.

### **Status bar**

The status bar can display the following information:

- Selected objects
- Action in progress
- User dialogs
- User entry prompts
- Requests for confirmation

#### **7.4.1 Selecting filters**

- Description** This function is not available in the user group “User”.  
The filter defines how programs are displayed in the file list. The following filters are available:
- **Detail**  
Programs are displayed as SRC and DAT files. (Default setting)
  - **Modules**  
Programs are displayed as modules.
- Precondition**
- “Expert” user group
- 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 group “Expert”.
- 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**.

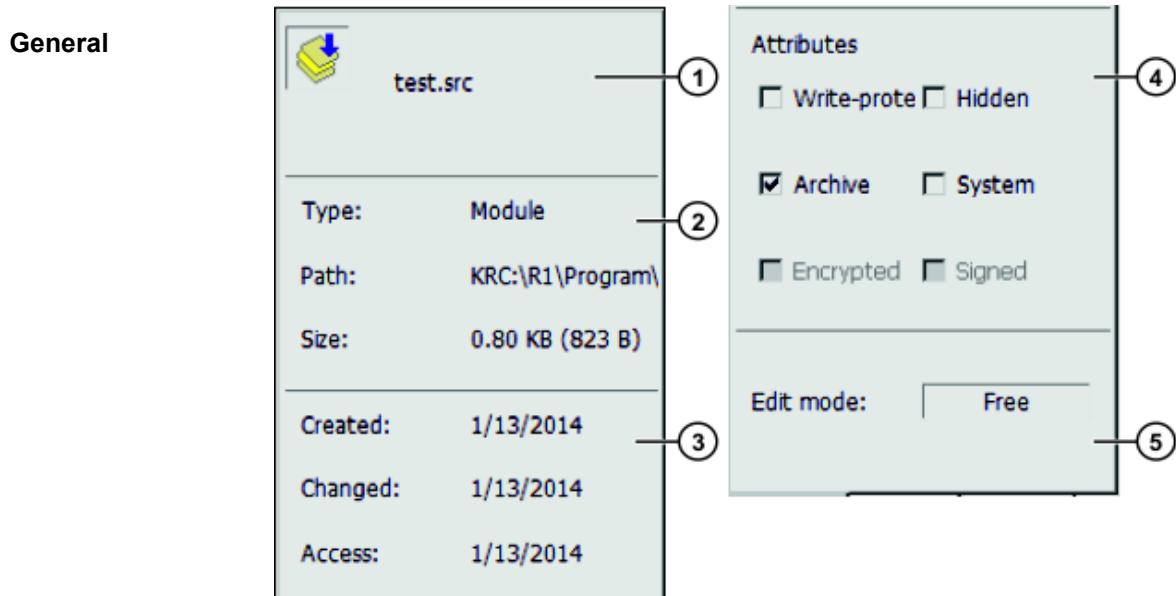


Fig. 7-2: "General" tab

Item	Description
1	Name of the selected object
2	Object type, path and size. Object types: <ul style="list-style-type: none"> <li>■ <b>Module:</b> module</li> <li>■ <b>Dir:</b> folder</li> <li>■ <b>Archiv:</b> archive file</li> <li>■ <b>Bin:</b> binary file</li> <li>■ <b>Text:</b> text file</li> <li>■ <b>VirtualDir:</b> virtual folder</li> <li>■ <b>Unknown:</b> all other file types</li> </ul>
3	Windows object properties
4	Windows object properties. The properties can be modified in the user group "Expert".
5	<b>Free:</b> the file is not selected in the smart.HMI and is not open. <b>Full:</b> the file is open in the smart.HMI. <b>ProKor:</b> the file is selected in the smart.HMI.

**Module info**

The **Module info** tab is only displayed if the selected object is a file.

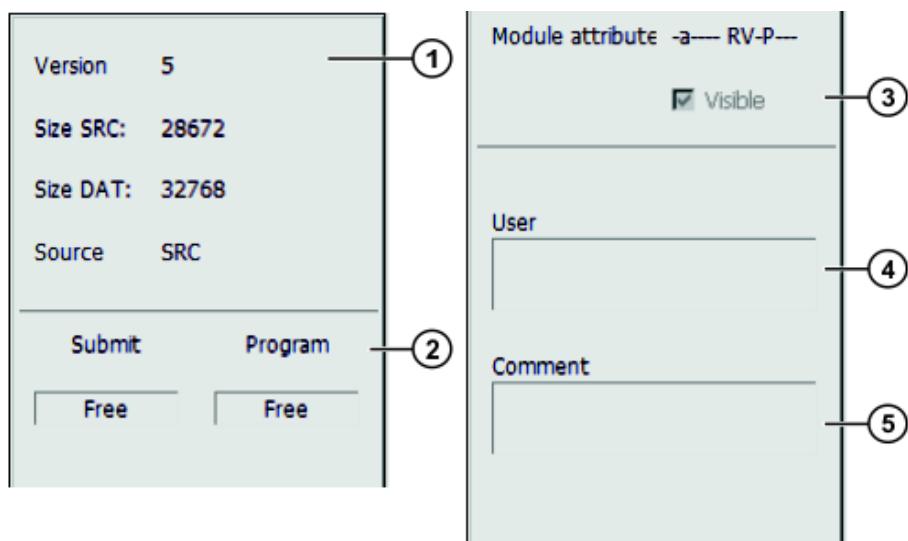


Fig. 7-3: “Module info” tab

Item	Description
1	<b>Version:</b> 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. <b>Size SRC::</b> size of the SRC file <b>Size DAT::</b> size of the DAT file <b>Source type::</b> file type <ul style="list-style-type: none"> <li>■ <b>SRC:</b> SRC file</li> <li>■ <b>SubmitSub:</b> SUB file</li> <li>■ <b>None:</b> all other file types, e.g. DAT file</li> </ul>
2	Status of the module in the submit interpreter and in the robot interpreter <b>Free:</b> program is not selected. <b>selected:</b> program is selected. <b>Active:</b> only relevant for the <b>Submit</b> box. This program is currently being used by the submit interpreter.
3	Check box active: if this program is called as a subprogram, it is displayed in the Editor. Check box not active: if this program is called as a subprogram, it is not displayed in the Editor. This program cannot be selected manually.
4	The user can enter his or her name here.
5	The user can enter a comment for the module here. The comment is displayed in the <b>Comment</b> column in the Navigator.

#### Parameters

The **Parameters** tab is only displayed if the selected object is a file.

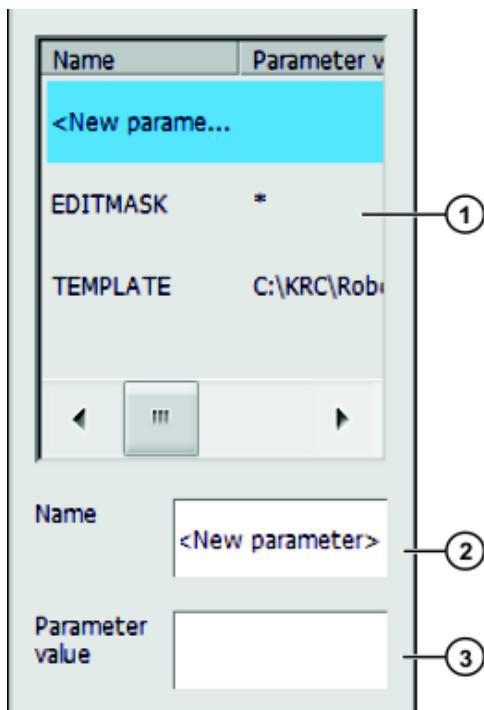


Fig. 7-4: “Parameters” tab

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 <b>Parameter value</b> . Then confirm with <b>OK</b> .
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 <b>Module info</b> , check box <b>Visible</b> <ul style="list-style-type: none"> <li>■ &amp;ACCESS RV = check box active</li> <li>■ &amp;ACCESS R = check box inactive</li> </ul>
2	Tab <b>Module info</b> , box <b>Version</b>
3	Tab <b>Module info</b> , box <b>Comment</b>
4	Tab <b>Module info</b> , box <b>User</b>
5	Tab <b>Parameters</b> , boxes <b>Name</b> and <b>Parameter value</b>

## 7.5 Selecting or opening a program

<b>Overview</b>	<p>A program can be selected or opened. Instead of the Navigator, an editor is then displayed with the program.</p> <p>(&gt;&gt;&gt; 7.5.1 "Selecting and deselecting a program" Page 243)</p> <p>(&gt;&gt;&gt; 7.5.2 "Opening a program" Page 244)</p> <p>It is possible to toggle backwards and forwards between the program display and the Navigator.</p> <p>(&gt;&gt;&gt; 7.5.3 "Toggling between the Navigator and the program" Page 245)</p>
<b>Differences</b>	<p><b>Program is selected:</b></p> <ul style="list-style-type: none"> <li>■ The block pointer is displayed.</li> <li>■ The program can be started.</li> <li>■ The program can be edited to a certain extent.</li> </ul> <p>Selected programs are particularly suitable for editing in the user group "User".</p> <p>Example: KRL instructions covering several lines (e.g. LOOP ... END-LOOP) are not permissible.</p> <ul style="list-style-type: none"> <li>■ When the program is deselected, modifications are accepted without a request for confirmation. If impermissible modifications are programmed, an error message is displayed.</li> </ul> <p><b>Program is open:</b></p> <ul style="list-style-type: none"> <li>■ The program cannot be started.</li> <li>■ The program can be edited.</li> </ul> <p>Opened programs are particularly suitable for editing in the user group "Expert".</p> <ul style="list-style-type: none"> <li>■ A request for confirmation is generated when the program is closed. Modifications can be accepted or rejected.</li> </ul>

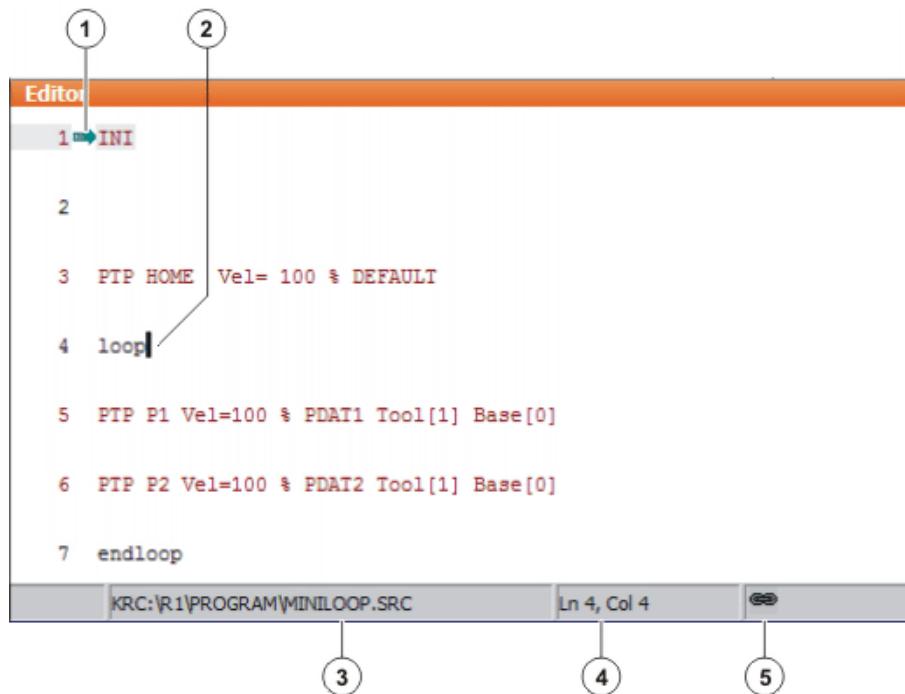
### 7.5.1 Selecting and deselecting a program

<b>Precondition</b>	<ul style="list-style-type: none"> <li>■ T1, T2 or AUT mode</li> </ul>
<b>Procedure</b>	<ol style="list-style-type: none"> <li>1. Select the program in the Navigator and press <b>Select</b>. 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.</li> <li>2. Start or edit the program.</li> <li>3. Deselect the program again: Select <b>Edit &gt; Cancel program</b>. or: In the status bar, touch the <b>Robot interpreter</b> status indicator. A window opens. Select <b>Cancel program</b>.</li> </ol>



When the program is deselected, modifications are accepted without a request for confirmation!

<b>Description</b>	<p>If a program is selected, this is indicated by the <b>Robot interpreter</b> status indicator.</p> <p>(&gt;&gt;&gt; 8.6 "Robot interpreter status indicator" Page 273)</p>
--------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------



**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

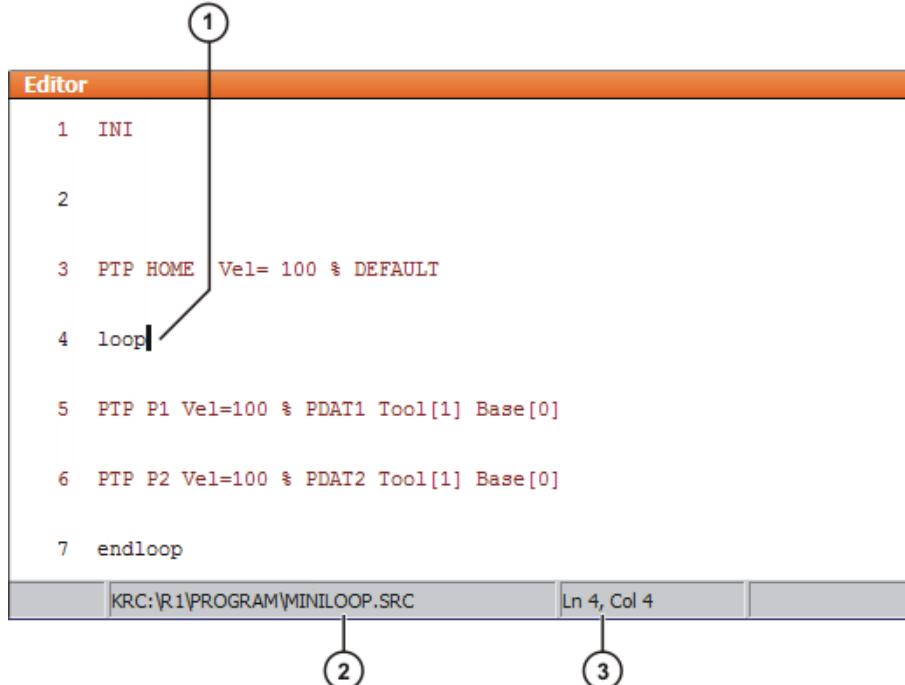
#### Precondition

- 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 247)
2	The INI line contains initializations for internal variables and parameters.
4	HOME position (>>> 7.6.1 "HOME position" Page 246)
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.



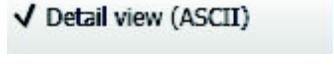
If the HOME position is modified, this affects all programs in which it is used. Injuries or damage to property may result.

## 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 group “Expert”
  - Program is selected or open.
- Procedure**
1. Select the menu sequence **Edit > View**. The subitem **DEF line** displays the current status:
    - **Check box not active**: The DEF line is hidden.
    - **Check box active**: The DEF line is displayed.
  2. To change the status, touch the menu item **DEF line**.  
The menu then closes automatically.

### 7.7.2 Activating detail view

- Description** Detail view (ASCII mode) is deactivated by default to keep the program transparent. If detail view is activated, hidden program lines, such as the FOLD and ENDFOLD lines and the DEF line, are displayed.
- Detail view is activated and deactivated separately for opened and selected programs.
- Precondition**
- “Expert” user group
- Procedure**
1. Select the menu sequence **Edit > View**. The subitem **Detail view (ASCII)** displays the current status:
    - **Check box not active**: Detail view is deactivated.
    - **Check box active**: Detail view is activated.
  2. To change the status, touch the menu item **Detail view (ASCII)**.  
The menu then closes automatically.

### 7.7.3 Activating/deactivating the line break function

- Description** If a line is wider than the program window, the line is broken by default. The part of the line after the break has no line number and is marked with a black, L-shaped arrow.

```
8 EXT IBGN {IBGN_COMMAND :IN,BOOL :IN,REAL :IN,REAL  
↓ :IN,BOOL :IN,E6POS :OUT }
```

Fig. 7-7: Line break

The line break function can be deactivated. If a line is wider than the program window, the line is no longer visible in its entirety. A scroll bar is displayed underneath the program window.

The line break function is activated and deactivated separately for opened and selected programs.

#### Precondition

- User group “Expert”
- 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.

 Line break

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.

- In the user group “User”, Folds are always closed. In other words, the contents of the Folds are not visible and cannot be edited.
- In the user group “Expert”, Folds are closed by default. They can be opened and edited. New Folds can be created.

(>>> 7.8.2 "Creating folds" Page 251)

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**
- User group “Expert”
  - 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

- Overview**
- A running program cannot be edited.
  - Programs cannot be edited in AUT EXT mode.

Action	Possible in user group ...?
Insert comment or stamp	<b>User:</b> Yes <b>Expert:</b> Yes
Create folds	<b>User:</b> No <b>Expert:</b> Yes
Cut	<b>User:</b> No <b>Expert:</b> Yes
Copy	<b>User:</b> No <b>Expert:</b> Yes

Action	Possible in user group ...?
Insert blank lines (press the Enter key)	<b>User:</b> No <b>Expert:</b> Yes
Paste	<b>User:</b> No <b>Expert:</b> Yes
Delete program lines	<b>User:</b> Yes <b>Expert:</b> Yes
Search	<b>User:</b> Yes <b>Expert:</b> Yes Possible for all user groups in an open program, even in AUT EXT mode.
Replace	<b>User:</b> No <b>Expert:</b> Yes (program is open, not selected)
Mark region	<b>User:</b> No <b>Expert:</b> Yes
Programming with inline forms	<b>User:</b> Yes <b>Expert:</b> Yes
KRL programming	<b>User:</b> Possible to a certain extent. KRL instructions covering several lines (e.g. LOOP ... END-LOOP) are not permissible. <b>Expert:</b> Yes

### 7.8.1 Inserting a comment or stamp

#### Precondition

- Program is selected or open.
- Operating mode T1

#### Procedure

1. Select the line after which the comment or stamp is to be inserted.
2. Select the menu sequence **Commands > Comment > Normal or Stamp**.
3. Enter the desired data. If a comment or stamp has already been entered previously, the inline form still contains the same entries.
  - In the case of a comment, the box can be cleared using **New text** ready for entry of a new text.
  - In the case of a stamp, the system time can also be updated using **New time** and the **NAME** box can be cleared using **New name**.
4. Save with **Cmd Ok**.

#### Description Comment

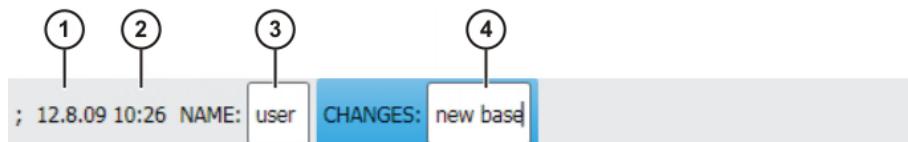


Fig. 7-10: Inline form “Comment”

Item	Description
1	Any text

#### Description Stamp

A stamp is a comment that is extended to include the system date and time and the user ID.



**Fig. 7-11: Inline form “Stamp”**

Item	Description
1	System date (cannot be edited)
2	System time
3	Name or ID of the user
4	Any text

### 7.8.2 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

- “Expert” user group
- Program is selected or open.
- Operating mode T1

#### Procedure

1. Enter Fold in program. A double semicolon prevents the Fold from closing when edited.

```

4
5  ;;FOLD outputs
6  $OUT[1] = TRUE
7  $OUT[2] = TRUE
8  ;;ENDFOLD outputs
9

```

**Fig. 7-12: Creating a sample Fold, step 1**

2. Delete the second semicolon.

```

4
5  ;FOLD outputs
6  $OUT[1] = TRUE
7  $OUT[2] = TRUE
8  ;ENDFOLD outputs
9

```

**Fig. 7-13: Creating a sample Fold, step 2**

3. Position the cursor in a line outside the Fold. The Fold closes.

```

4
5  outputs
6

```

**Fig. 7-14: Creating a sample Fold, step 3**

### 7.8.3 Selecting a range

**Precondition**

- User group “Expert”
- Program is selected or open.

**Procedure**

1. Touch the line in which the selection is to start.
2. Select the menu sequence **Edit > Mark region**.  
The line in which the cursor is positioned is now selected.
3. Touch the line in which the selection is to end.  
The range between the lines, including the start and end lines, is now selected.

### 7.8.4 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 or range that is to be deleted.  
If only one line is to be deleted, it does not have to have a color background. It is sufficient for the cursor to be positioned in the line.
2. Select the menu sequence **Edit > Delete**.
3. Confirm the request for confirmation with **Yes**.

### 7.8.5 Additional editing functions

The following additional program editing functions can be called using **Edit**:

**Copy**

Precondition:

- Program is selected or open.
- “Expert” user group
- Operating mode T1

**Paste**

Precondition:

- Program is selected or open.
- “Expert” user group
- Operating mode T1

**Cut**

Precondition:

- Program is selected or open.
- “Expert” user group
- Operating mode T1

**Find**

Precondition:

- Program is selected or open.

#### **Replace**

Precondition:

- Program has been opened.
- “Expert” user group
- Operating mode T1

#### **Marked region**

(>>> 10.6.4 "Transforming blocks of coordinates" Page 343)

## **7.9 Printing a program**

### **Procedure**

1. Select the program in the Navigator. Multiple program selection is also possible.
2. Select the menu sequence **Edit > Print**.

## **7.10 Archiving and restoring data**

### **7.10.1 Archiving overview**

**Target locations** Archiving can be performed to the following target destinations:

- USB stick in smartPAD or robot controller
- Network

### **Menu items**

The following menu items are available:

("\*.\*" means all files and subdirectories.)

<b>Menu item</b>	<b>Archives the directories/files</b>
<b>All</b>	<ul style="list-style-type: none"> <li>■ KRC:\*.*</li> <li>■ C:\KRC\Roboter\Config\User\*.*</li> <li>■ C:\KRC\Roboter\Config\System\Common\Mada\*.*</li> <li>■ C:\KRC\Roboter\Init\*.*</li> <li>■ C:\KRC\Roboter\Ir_Spec\*.*</li> <li>■ C:\KRC\Roboter\Template\*.*</li> <li>■ C:\KRC\Roboter\Rdc\*.*</li> <li>■ C:\KRC\User\*.*</li> <li>■ C:\KRC\Roboter\log\Mastery.log</li> <li>■ Some additional log data</li> </ul>
<b>Applications</b>	<ul style="list-style-type: none"> <li>■ KRC:\R1\Program\*.*</li> <li>■ KRC:\R1\System\*.*</li> <li>■ KRC:\R1\cell\*.*</li> <li>■ KRC:\Steu\\$config\*.*</li> </ul>

Menu item	Archives the directories/files
<b>System data</b>	<ul style="list-style-type: none"> <li>■ KRC:\R1\Mada\*.*</li> <li>■ KRC:\R1\System\*.*</li> <li>■ KRC:\R1\TP\*.*</li> <li>■ KRC:\Steu\Mada\*.*</li> <li>■ C:\KRC\Roboter\Config\User\*.*</li> <li>■ C:\KRC\Roboter\Config\System\Common\Mada\*.*</li> <li>■ C:\KRC\Roboter\Init\*.*</li> <li>■ C:\KRC\Roboter\Ir_Spec\*.*</li> <li>■ C:\KRC\Roboter\Template\*.*</li> <li>■ C:\KRC\Roboter\Rdc\*.*</li> <li>■ C:\KRC\User\*.*</li> </ul>
<b>Log data</b>	<ul style="list-style-type: none"> <li>■ C:\KRC\Roboter\log\*.*</li> </ul> <p>Except: Poslog.xlsx and files with the extension DMP</p> <ul style="list-style-type: none"> <li>■ Some additional log data</li> </ul>
<b>KrcDiag</b>	<p>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.</p> <p>A screenshot of the current view of the smartHMI is automatically generated for the data 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.</p> <p>In addition to the menu sequence <b>File &gt; Archive</b>, there are other methods available for compressing these data.</p> <p>(&gt;&gt;&gt; 13.5 "Automatically compressing data for error analysis (KrcDiag)" Page 486)</p>

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 robot name with that in the archive. If the names are different, a request for confirmation is generated.

If archiving is carried out repeatedly via **KrcDiag**, a maximum of 10 archives can be created. Further archives will overwrite the oldest existing archive.

The logbook can also be activated. (>>> 7.10.4 "Archiving the logbook" Page 255)

### 7.10.2 Archiving to a USB stick

#### Description

This procedure generates a ZIP file on the stick. By default, this file has the same name as the robot. A different name can be defined for the file, however, under **Start-up > Robot data**.

The archive is displayed in the ARCHIVE:\ directory in the Navigator. Archiving is also carried out automatically to D:\ as well as to the stick. The file INTERN.ZIP is generated here.

Special case **KrcDiag**:

This menu item generates the folder **KRCDiag** on the stick. This contains a ZIP file. The ZIP file is also automatically archived in C:\KUKA\KRCDiag.

**NOTICE**

A non-bootable USB stick must be used.  
We recommend using a non-bootable KUKA stick. Data may be lost if a stick from a different manufacturer is used.

**Procedure**

1. Connect the USB stick (to smartPAD or cabinet).
2. In the main menu, select **File > Archive > USB (KCP)** or **USB (cabinet)** and then the desired menu item.
3. Confirm the request for confirmation with **Yes**. The archive is created.  
Once the archiving is completed, this is indicated in the message window.  
**Special case KrcDiag:** If archiving is carried out using this menu item, a separate window indicates when archiving has been completed. The window is then automatically hidden again.
4. The stick can now be removed.

**7.10.3 Archiving on the network****Description**

This procedure generates a ZIP file on the network path. By default, this file has the same name as the robot. A different name can be defined for the file, however, under **Start-up > Robot data**.

The network path to which the data are to be archived must be configured under **Start-up > Robot data**. If a user name and password are required for archiving to this path, these can also be entered here.

The archive is displayed in the ARCHIVE:\ directory in the Navigator. Archiving is also carried out automatically to D:\ as well as to the network path. The file INTERN.ZIP is generated here.

**Special case KrcDiag:**

This menu item generates the folder **KRCDiag** on the network path. This contains a ZIP file. The ZIP file is also automatically archived in C:\KUKA\KRCDiag.

**Precondition**

- The network path to which the data are to be archived is configured.

**Procedure**

1. In the main menu, select **File > Archive > Network** and then the desired menu item.
2. Confirm the request for confirmation with **Yes**. The archive is created.  
Once the archiving is completed, this is indicated in the message window.  
**Special case KrcDiag:** If archiving is carried out using this menu item, a separate window indicates when archiving has been completed. The window is then automatically hidden again.

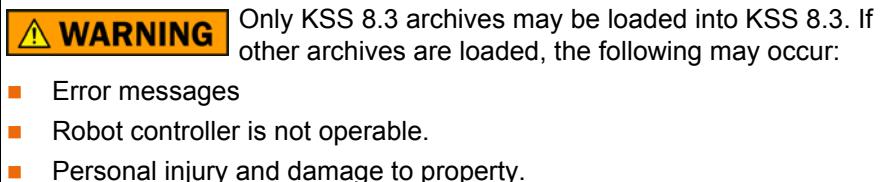
**7.10.4 Archiving the logbook****Description**

The file "Logbuch.txt" is generated as an archive in the directory C:\KRC\ROBOTER\LOG.

**Procedure**

- In the main menu, select **File > Archive > Logbook**.  
The archive is created. Once the archiving is completed, this is indicated in the message window.

### 7.10.5 Restoring data

**Description**

The following menu items are available for restoring data:

- **All**
- **Applications**
- **System data**

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**

- If data are to be restored from the USB stick: A USB stick with the archive is connected.

The stick can be connected to the smartPAD or robot controller.

**NOTICE**

A non-bootable USB stick must be used.  
We recommend using a non-bootable KUKA stick. Data may be lost if a stick from a different manufacturer is used.

**Procedure**

1. In the main menu, select **File > Restore** and then the desired subitems.
2. Confirm the request for confirmation with **Yes**. Archived files are restored to the robot controller. A message indicates completion of the restoration process.
3. If data have been restored from a USB stick: the stick can now be removed.
4. Reboot the robot controller.

## 7.11 Project management

### 7.11.1 Pinning a project on the robot controller

**Description**

Projects that are present on the robot controller can be pinned. A project can be pinned directly on the robot controller or in WorkVisual.

Pinned projects cannot be changed, activated or deleted. They can be copied or unpinned, however. A project can thus be pinned, e.g. to prevent it from being accidentally deleted.



Information about how projects can be pinned via WorkVisual can be found in the **WorkVisual** documentation.

**Precondition**

- “Expert” user group

**Procedure**

1. Touch the WorkVisual icon on the smartHMI, then go to **Open**. The **Project management** window opens.
2. Select the desired project and press the **Pin** button. The project is pinned and labeled with a pin symbol in the project list.  
(>>> 7.11.3 "Project management window" Page 258)

3. The project can be unpinned again by pressing the **Unpin** button.

### 7.11.2 Activating a project

- Precondition**
- User group “Expert” or higher  
If the activation would cause changes in the area **Safety-relevant communication parameters**, the user group “Safety recovery” or higher must be selected.
  - In AUT or AUT EXT mode (KSS) or in EXT mode (VSS):  
The project can only be activated if this affects only KRL programs. If the project contains settings that would cause other changes, it cannot be activated.



If one of the options KUKA.SafeOperation or KUKA.SafeRangeMonitoring is installed on the robot controller, different user groups may apply. Information can be found in the documentation for these options.

**Preparation**

There are 2 ways of reaching the first step of the procedure below.

- Project activation is the direct continuation of another sequence, e.g. the restoration of a project.  
In this case, the preparation described as follows is not necessary.
- Or: Project activation is executed as a stand-alone sequence.  
In this case, the preparation described as follows is required in order to arrive at the procedure.

Preparation:

1. Touch the WorkVisual icon on the smartHMI, then go to **Open**. The **Project management** window opens.
2. Select the desired project and activate it using the **Activate** button.

**Procedure**

1. 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.  
If no relevant project will be overwritten: Confirm with **Yes** within 30 minutes.
2. 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.



If changes are listed in the overview under the heading **Safety-relevant communication parameters**, this means that the behavior of the Emergency Stop and “Operator safety” signal may have changed compared with the previous project.  
After activation of the project, the Emergency Stop and the “Operator safety” signal must be checked for safe functioning. If the project is activated on several robot controllers, this check must be carried out for every robot controller. Failure to carry out this check may result in death, injuries or damage to property.

3. The overview displays the request for confirmation *Do you want to continue?*. Confirm with **Yes**. The project is activated on the robot controller.

**WARNING**

After activation of a project on the robot controller, the safety configuration must be checked there! If this is not done, the robot will possibly be operated with incorrect data. Death, injuries or damage to property may result.

(>>> 6.6 "Checking the safety configuration of the robot controller"  
Page 171)

**WARNING**

If the activation of a project fails, an error message is displayed. In this case, one of the following measures must be carried out:

- Either: Activate a project again (the same one or a different one).
- Or: Reboot the robot controller with a cold restart.



In the case of a KSS/VSS update, the initial project and base project are overwritten by copies of the active project.

### 7.11.3 Project management window

#### Overview

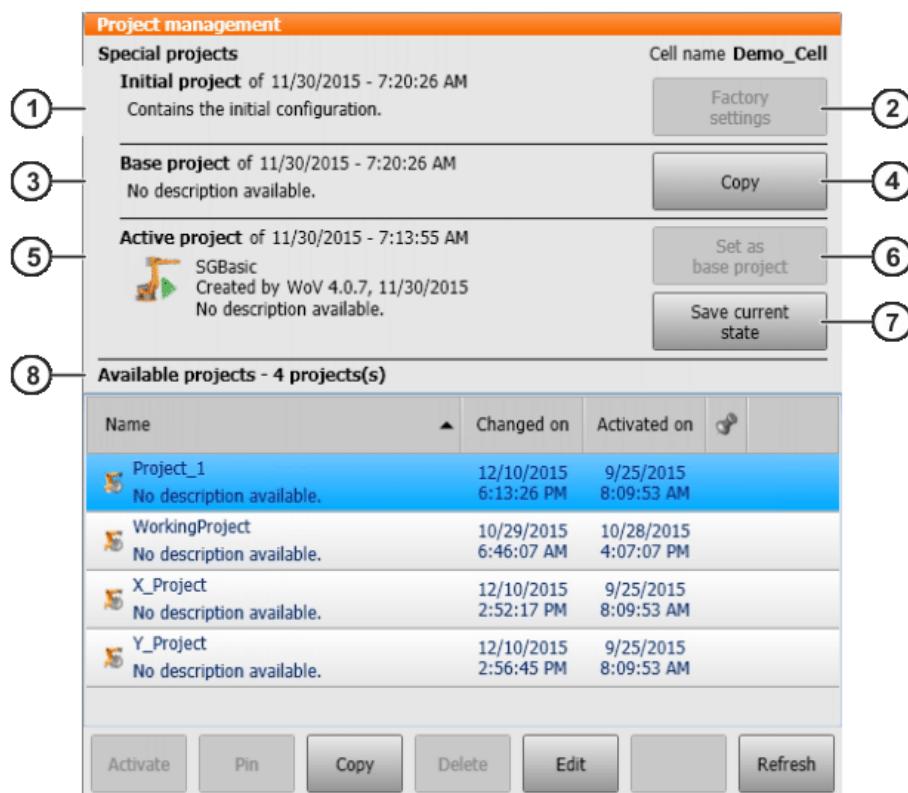
The **Project management** window is opened using the WorkVisual icon on the smartHMI.

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	<p>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.</p> <p>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).</p> <p>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.)</p>



In the case of a KSS/VSS update, the initial project and base project are overwritten by copies of the active project.

**Description****Fig. 7-15: “Projects management” window**

Item	Description
1	The initial project is displayed.
2	Restores the factory settings of the robot controller. Only available to the user group “Expert” or higher.
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. Only available to the user group “Expert” or higher.
7	Creates a pinned copy of the active project.
8	List of inactive projects (except base and initial project)

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
<b>Activate</b>	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.  Only available to the user group "Expert" or higher.
<b>Pin</b>	Pins the project.  (>>> 7.11.1 "Pinning a project on the robot controller" Page 256)  Only available if an unpinned project is selected. Only available to the user group "Expert" or higher.
<b>Unpin</b>	Unpins the project.  Only available if a pinned project is selected. Only available to the user group "Expert" or higher.
<b>Copy</b>	Copies the selected project.
<b>Delete</b>	Deletes the selected project.  Only available if a non-activated, unpinned project is selected. Only available to the user group "Expert" or higher.
<b>Edit</b>	Opens a window in which the name and/or description of the selected project can be changed.
<b>Refresh</b>	Refreshes the project list. This enables, for example, projects to be displayed which have been transferred to the robot controller since the display was opened.

## 7.12 Backup Manager

### 7.12.1 Overview of Backup Manager

**Overview** The Backup Manager makes it possible to back up and restore projects, option packages and RDC data.

The default settings for the Backup Manager are defined in the **Backup configuration** window. The settings can be changed.

#### Start types

There are several ways of starting a backup or restoration:

	Backup	Restoration
<b>Manual</b>	Yes	Yes
<b>Automatic, in intervals</b>	Yes	No
<b>Via inputs</b>  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.

	<b>Execution</b>
<b>Manual</b>	(>>> 7.12.2 "Backing up projects, option packages and RDC data manually" Page 261) (>>> 7.12.3 "Manually restoring projects and option packages" Page 262) (>>> 7.12.4 "Restoring RDC data manually" Page 263)
<b>Automatic, in intervals</b>	If backups are to be started automatically, this must be configured in the <b>Backup configuration</b> window.
<b>Via inputs</b>	If backups/restorations are to be started via inputs, this must be configured in the <b>Backup configuration</b> window.

## 7.12.2 Backing up projects, option packages and RDC data manually

- Projects** The following projects are backed up by default:
- Active project
  - Initial project
  - Base project
- Option packages** Option packages will be backed up under the following conditions:
- The option package has a KOP file.
  - The option package was originally added to the project in WorkVisual. The project is now active on the robot controller.
- Or:
- The option package has been installed in the active project via **Start-up > Additional software**. The option package was available during installation as a single KOP file (not as a directory structure!).
- RDC data** Every time a backup is made, a file with the name *[Robot\_serial\_number].RDC* is created. It contains the CAL, MAM and PID files. Not all files are present in all cases (dependent on the robot).
- Menu items** There are 2 menu items available for backup. They differ with regard to the target paths.
- **Back up**  
Backup is carried out to the paths stored in the following boxes in the **Backup configuration** window: **Target path for project backup** and **Target path for KOP backup**.
  - **Save as...**  
A path can be set here. It only applies to this backup. The option packages are also backed up to this path by default. If necessary, however, a separate path can be set for the option packages.
- If the paths do not yet exist, they are automatically created during the backup operation.
- Procedure**
- Back up:**
- In the main menu, select **File > Backup Manager > Back up**.
- Save as... :**
1. In the main menu, select **File > Backup Manager > Save as....**
  2. Enter the target in the **Target path for project backup** box.

3. If necessary, enter a separate path for the option packages.  
To do so, activate the check box **Divert path for option packages** and enter the desired path in the **Target path for KOP backup** box.
4. Start the backup with **Save**.

The backup is carried out. The robot controller displays messages when the backup has been successfully completed. It generates one message per project or option package and one message relating to the RDC data.

However, option packages will not be backed up if the same package version already exists in the target directory.

### 7.12.3 Manually restoring projects and option packages

#### Projects

The following projects are restored by default:

- Active project (i.e. the project which was active during the backup)
- Initial project
- Base project

After manual restoration of the project, the robot controller reactivates the previously active project if certain preconditions are met. If not, the project remains inactive. This can be activated by the user at the desired point in time.

After restoration of the project via inputs, the previously active project also remains inactive.

#### Option packages

The option packages for which there are KOP files in the source directory are not necessarily all restored. An option package will be restored under the following conditions:

- At the time of backup, the option package was present in the active project.
- At the time of restoration, this version of the option package is not installed on the robot controller.

Following restoration, some option packages automatically install themselves when the corresponding project is activated.

Option packages that do not automatically install themselves are available for installation under **Start-up > Additional software**. A message on the robot controller indicates that the option package must still be installed.

#### Menu items

There are 2 menu items available for restoration. They differ with regard to the paths that are accessed.

- **Restore**

The paths stored in the following boxes in the **Backup configuration** window are accessed: **Target path for project backup** and **Target path for KOP backup**.

- **Restore from ...**

A path can be set here. It only applies to this restoration. If necessary, a separate path can be set for the option packages.

#### Precondition



No special preconditions are required for the restoration as such. If a project is to be activated, however, the following preconditions must be met.

- User group "Expert" or higher

If the activation would cause changes in the area **Safety-relevant communication parameters**, the user group "Safety recovery" or higher must be selected.

- In AUT or AUT EXT mode (KSS) or in EXT mode (VSS):  
The project can only be activated if this affects only KRL programs. If the project contains settings that would cause other changes, it cannot be activated.



If one of the options KUKA.SafeOperation or KUKA.SafeRangeMonitoring is installed on the robot controller, different user groups may apply. Information can be found in the documentation for these 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 (KSS) or EXT (VSS):  
The robot controller starts to activate the project without a request for confirmation.

For the further activation steps, see: ([>>> 7.11.2 "Activating a project"](#)  
Page 257)



Observe the safety instructions relating to project activation!

## 7.12.4 Restoring RDC data manually

### Menu items

There are 2 menu items available for restoration. They differ with regard to the path that is accessed.

- **Restore**  
The path stored under **Target path for project backup** in the **Backup configuration** window is accessed.
- **Restore from ...**  
A path can be set here. It only applies to this restoration.

### Precondition

- “Expert” user group

**Procedure****Restore:**

- In the main menu, select **File > Backup Manager > Restore > RDC data.**

**Restore from ... :**

1. In the main menu, select **File > Backup Manager > Restore from ... > RDC data.**
2. Select the source path.
3. A window opens. Activate the check boxes next to the data that are to be restored:
  - **PID file, MAM file and/or CAL file**If an entry is grayed out, this means that this file is not available in the backup.
4. Confirm the selection with **Restore**.

Once the restoration has been completed, the following message is displayed:  
*RDC data successfully restored from {0}.*

**7.12.5 Configuring Backup Manager****Precondition**

- “Expert” user group
- For changes to the **Signal interface** tab additionally:  
T1 or T2 mode

**Procedure**

1. In the main menu, select **File > Backup Manager > Backup configuration.**  
The **Backup configuration** window is opened.
2. Carry out the desired changes in the tabs.
  - **Backup configuration** contains the general settings.  
Automatic saving can also be configured here if required.
  - I/O control can also be configured under **Signal interface** if required.
3. Close the window.
4. Respond to the request for confirmation asking whether the changes should be saved by pressing **Yes**.

### 7.12.5.1 Backup configuration tab

#### Backup configuration

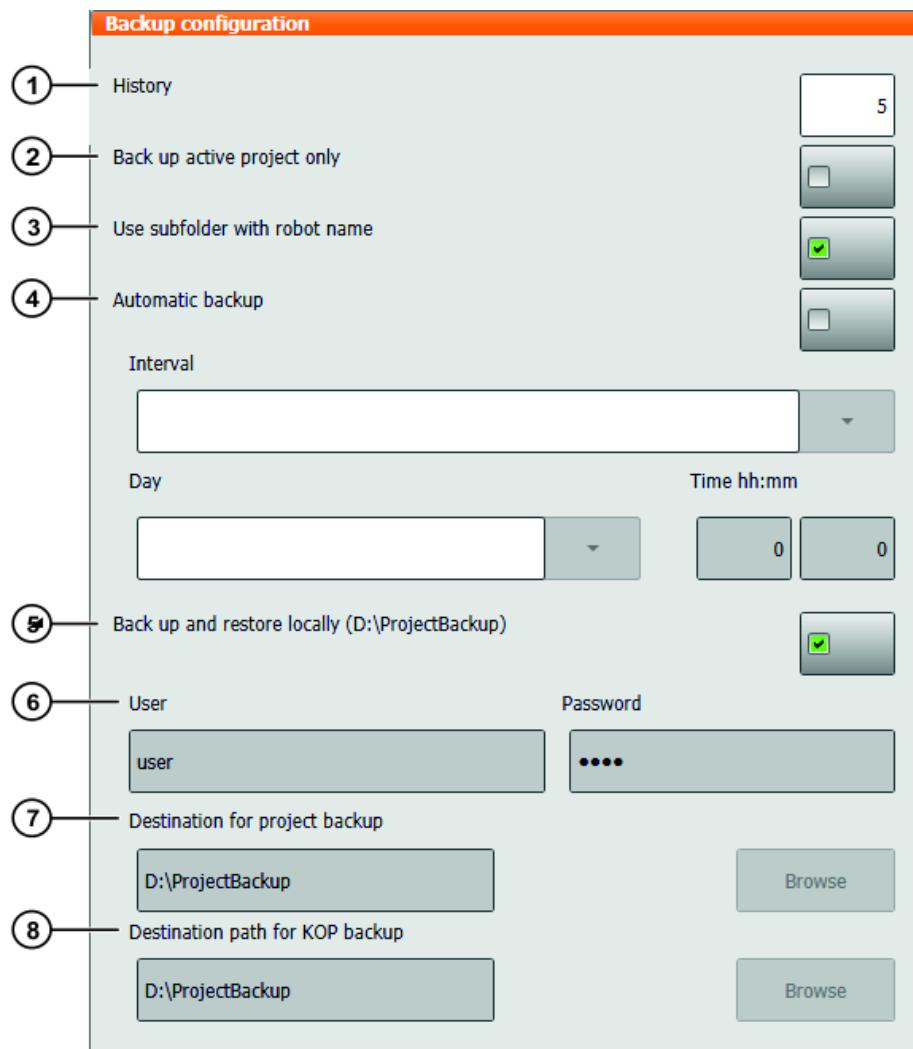


Fig. 7-16: "Backup configuration" tab

Item	Description
1	<p>Maximum number of subfolders for older backups</p> <p>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.</p> <p>■ <b>0 ... 50</b></p> <p>Default: 5</p>
2	<ul style="list-style-type: none"> <li>■ <b>Activated:</b> The active project is backed up during backup. During restoration, only the project which was active during the backup will be restored.</li> <li>■ <b>Deactivated</b> (default): The following projects are backed up during the backup: Active project, base project, initial project. These projects are restored during restoration.</li> </ul>

Item	Description
3	<ul style="list-style-type: none"> <li>■ <b>Active</b> (default): During backup, the robot controller stores the projects and RDC data in the path [<b>Target path for project backup</b>]\[Robot name]\.</li> </ul> <p>The robot controller accesses this path during restoration.</p> <p><b>Note:</b> Select this setting if more than one robot controller uses the same target path.</p> <ul style="list-style-type: none"> <li>■ <b>Inactive</b>: During backup, the robot controller stores the projects and RDC data in the path [<b>Target path for project backup</b>]\.</li> </ul> <p>The robot controller accesses this path during restoration.</p> <p><b>Note:</b> Option packages are always stored under [<b>Target path for KOP backup</b>]\OptionPackages\ irrespective of this setting.</p>
4	<ul style="list-style-type: none"> <li>■ <b>Active</b>: The robot controller automatically carries out backups. The following parameters determine the point in time: <b>Interval, Day, Time hh:mm</b></li> </ul> <p><b>Note:</b> After an automatic backup, the robot controller does not display a message of successful completion.</p> <p>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.</p> <ul style="list-style-type: none"> <li>■ <b>Deactivated</b> (default): No automatic backup.</li> </ul>
5	<ul style="list-style-type: none"> <li>■ <b>Active</b> (default): The target path for backups and the source path for restorations is D:\ProjectBackup.</li> <li>■ <b>Inactive</b>: The following parameters determine the target path for backups and the source path for restorations: <b>Target path for project backup, Target path for KOP backup</b></li> </ul>
The following parameters can only be edited if the option <b>Back up and restore locally (D:\ProjectBackup)</b> is deactivated.	
6	<p>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.</p> <ul style="list-style-type: none"> <li>■ <b>User</b>: User name Default: user</li> <li>■ <b>Password</b>: Password. On entry, only the number of characters is displayed and not the characters themselves. Default: kuka</li> </ul>
7	<p>For projects and RDC data: Target path for backups and source path for restorations</p> <p>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.</p>
8	<p>For KOP files: Target path for backups and source path for restorations</p> <p>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.</p>

### 7.12.5.2 “Signal interface” tab

#### Signal interface

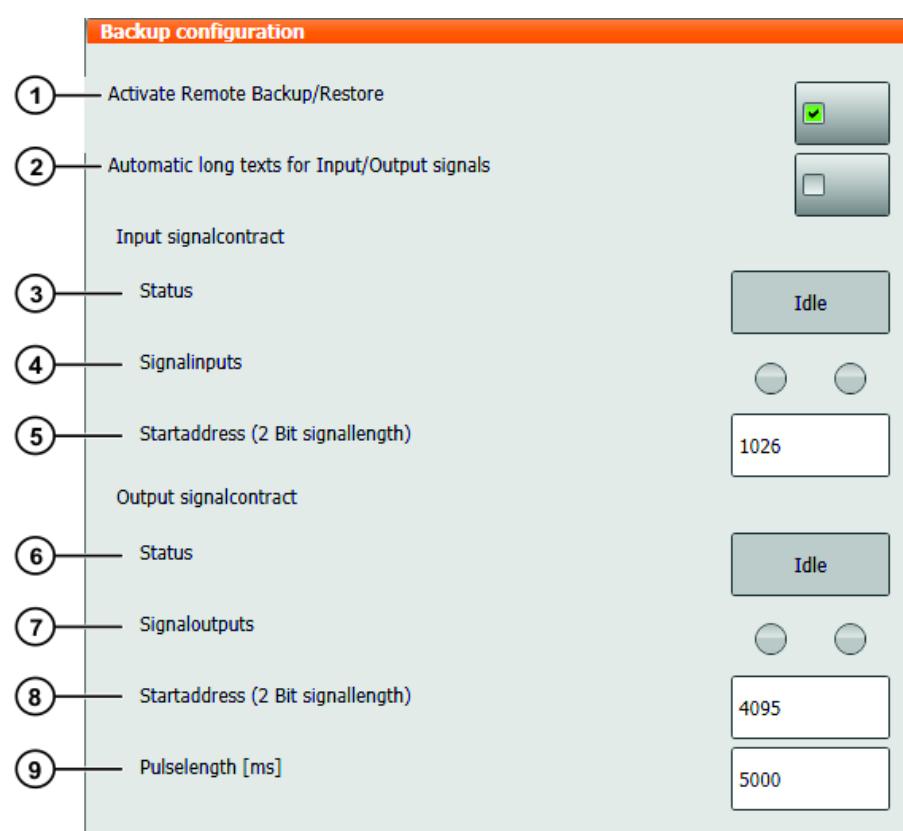


Fig. 7-17: “Signal interface” tab

Item	Description
1	<ul style="list-style-type: none"> <li><b>Activated:</b> Backups and restorations can be started via the input signal configured under <b>Start address</b>.</li> <li><b>Deactivated</b> (default): Start is not possible via input signal.</li> </ul>
The following parameters can only be edited if the option <b>Activate Remote Backup/Restore</b> is activated.	
2	<ul style="list-style-type: none"> <li><b>Activated:</b> The signals defined under <b>Start address</b> are assigned the following long text names: <b>BM input signal, BM output signal</b> If the signals already have long text names, these are not overwritten.</li> <li><b>Deactivated</b> (default): No long text names If <b>Activated</b> was previously selected, <b>Deactivated</b> removes the long text names.</li> </ul>
3	Displays when the input signal starts a backup or restoration. (Display during the length of a signal) (>>> "Input signals" Page 268)
4	<ul style="list-style-type: none"> <li>Left-hand lamp: Status of the input <b>Start address</b>.</li> <li>Right-hand lamp: Status of the input <b>Start address + 1</b>.</li> </ul> <p>Status (cannot be edited): Green = 1; Gray = 0</p>

Item	Description
5	Input signal consisting of the following inputs: <b>Start address</b> and <b>Start address + 1</b> . Default: 1 026
6	Displays the action being executed or the result. (>>> "Output signals" Page 268)
7	<ul style="list-style-type: none"> <li>■ Left-hand lamp: Status of the output <b>Start address</b>.</li> <li>■ Right-hand lamp: Status of the output <b>Start address + 1</b>.</li> </ul> Status (cannot be edited): Green = 1; Gray = 0
8	Output signal consisting of the following outputs: <b>Start address</b> and <b>Start address + 1</b> . 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 / <b>Idle</b>
01	Starts a backup / <b>Backup</b>
10	Starts a restoration / <b>Restore</b>
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 / <b>Idle</b>
Backup running	10 / <b>Backup</b>
Restoration running	10 / <b>Restore</b>
Backup successful	01 / <b>BackupSuccess</b>
Restoration successful	01 / <b>RestoreSuccess</b>
Backup unsuccessful	11 / <b>BackupError</b>
Restoration unsuccessful	11 / <b>RestoreError</b>

## 8 Program execution

### 8.1 Selecting the program run mode

- Procedure**
1. Touch the **Program run mode** status indicator. The **Program run mode** window is opened.
  2. Select the desired program run mode.
- The window closes and the selected program run mode is applied.

### 8.2 Program run modes

Designation	Status indicator	Description
<b>Go</b> #GO		The program is executed through to the end without stopping.
<b>Motion</b> #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.
<b>Single Step</b> #ISTEP		<p>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.</p> <p><b>Single Step</b> is only available to the “Expert” user group.</p>
<b>Backwards</b> #BSTEP		<p>This program run mode is automatically selected if the Start backwards key is pressed. It is not possible to select a different mode.</p> <p>This mode works in the same way as <b>Motion</b>, 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.</p> <p>This exception does not apply in the case of SCIRC motions. Here, the backward motion is always stopped at the auxiliary point.</p>

The following additional program run modes are available for systems integrators.

These program run modes can only be selected via the variable correction function. System variable for the program run mode: \$PRO\_MODE.

Designation	Status indicator	Description
<b>Program Step</b> #PSTEP		The program is executed step by step without advance processing. Subprograms are executed completely.
<b>Continuous Step</b> #CSTEP		Approximate positioning points are executed with advance processing, i.e. they are approximated.  Exact positioning points are executed without advance processing and with a stop after the motion instruction.

### 8.3 Advance run

The advance run is the maximum number of motion blocks that the robot controller calculates and plans in advance during program execution. The actual number is dependent on the capacity of the computer.

The advance run refers to the current position of the block pointer. It is set via the system variable \$ADVANCE:

- Default value: 3
- Maximum value: 5

The advance run is required, for example, in order to be able to calculate approximate positioning motions. If \$ADVANCE = 0 is set, approximate positioning is not possible.

Certain statements trigger an advance run stop. These include statements that influence the periphery, e.g. OUT statements.

### 8.4 Block pointer

#### Overview

During program execution, the block pointer indicates various items of information:

- Which motion the robot is currently executing or has completed
- Whether an auxiliary point or end point is currently being approached
- The direction in which the robot is executing the program

Pointer	Direction	Description
	Forwards	The end point is being approached.
	Backwards	
	Forwards	The end point has been reached with exact positioning.
	Backwards	
	Forwards	The auxiliary point is being approached.
	Backwards	
	Forwards	The auxiliary point has been reached with exact positioning.
	Backwards	

## Examples for forward motion

```

5 PTP P3 Vel=100 % PDAT1 Tool[1] Base[0]

6 ➔PTP P4 Vel=100 % PDAT2 Tool[1] Base[0]

7 PTP P5 Vel=100 % PDAT3 Tool[1] Base[0]

```

Fig. 8-1: 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. 8-2: 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. 8-3: The robot is moving from P5 to auxiliary point P6

```

6 PTP P5 Vel=100 % PDAT3 Tool[1] Base[0]

7 ➔CIRC P6 P7 Vel=2 m/s CPDAT1 Tool[1] Base[0]

8 PTP P8 Vel=100 % PDAT16 Tool[1] Base[0]

```

Fig. 8-4: The robot has reached auxiliary point P6 with exact positioning

```

6 PTP P5 Vel=100 % PDAT3 Tool[1] Base[0]

7 ➔CIRC P6 P7 Vel=2 m/s CPDAT1 Tool[1] Base[0]

8 PTP P8 Vel=100 % PDAT16 Tool[1] Base[0]

```

Fig. 8-5: The robot is moving from auxiliary point P6 to P7

```

6 PTP P5 Vel=100 % PDAT3 Tool[1] Base[0]

7 ➔CIRC P6 P7 Vel=2 m/s CPDAT1 Tool[1] Base[0]

8 PTP P8 Vel=100 % PDAT16 Tool[1] Base[0]

```

Fig. 8-6: The robot has reached P7 with exact positioning

**Examples for  
backward motion**

```

6 PTP P5 Vel=100 % PDAT3 Tool[1] Base[0]

7 CIRC P6 P7 Vel=2 m/s CPDAT1 Tool[1] Base[0]

8 ↑ PTP P8 Vel=100 % PDAT16 Tool[1] Base[0]

```

Fig. 8-7: The robot is moving from P8 to P7

```

6 PTP P5 Vel=100 % PDAT3 Tool[1] Base[0]

7 → CIRC P6 P7 Vel=2 m/s CPDAT1 Tool[1] Base[0]

8 PTP P8 Vel=100 % PDAT16 Tool[1] Base[0]

```

Fig. 8-8: The robot has reached P7 with exact positioning

```

6 PTP P5 Vel=100 % PDAT3 Tool[1] Base[0]

7 ↑ CIRC P6 P7 Vel=2 m/s CPDAT1 Tool[1] Base[0]

8 PTP P8 Vel=100 % PDAT16 Tool[1] Base[0]

```

Fig. 8-9: The robot is moving from P7 to auxiliary point P6

```

6 PTP P5 Vel=100 % PDAT3 Tool[1] Base[0]

7 → CIRC P6 P7 Vel=2 m/s CPDAT1 Tool[1] Base[0]

8 PTP P8 Vel=100 % PDAT16 Tool[1] Base[0]

```

Fig. 8-10: The robot has reached auxiliary point P6 with exact positioning

**Double  
upward/downwar  
d arrow**

If the program window shows a section in which the block pointer is not currently located, a double arrow indicates the direction in which it is to be found.

```

↑
7 PTP P6 Vel=100 % PDAT4 Tool[1] Base[0]

8 PTP P7 Vel=100 % PDAT5 Tool[1] Base[0]

```

Fig. 8-11: The block pointer is located higher up in the program

```

14 PTP P13 Vel=100 % PDAT11 Tool[1] Base[0]

15 PTP P14 Vel=100 % PDAT12 Tool[1] Base[0]
↓

```

Fig. 8-12: The block pointer is located lower down in the program

## 8.5 Setting the program override (POV)

- 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.
- Procedure**
1. Touch the **POV/HOV** status indicator. The **Overrides** window is opened.
  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 value can be set to 100 %, 75 %, 50 %, 30 %, 10 %, 3 %, 1 %
    - Slider: The override can be adjusted in 1 % steps.
  3. Touch the **POV/HOV** status indicator again. (Or touch the area outside the window.)
- The window closes and the selected override value is applied.



The **Jog options** window can be opened via **Options** in the **Overrides** window.

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

## 8.6 Robot interpreter status indicator

Icon	Color	Description
	Gray	No program is selected.
	Yellow	The block pointer is situated on the first line of the selected program.
	Green	The program is selected and is being executed.
	Red	The selected and started program has been stopped.
	Black	The block pointer is situated at the end of the selected program.

## 8.7 Starting a program forwards (manual)

- Precondition**
- A program is selected.
  - Operating mode T1 or T2
- Procedure**
1. Select the program run mode.
  2. Hold the enabling switch down and wait until the status bar indicates "Drives ready":



Fig. 8-13

3. Carry out a BCO run: Press Start key and hold it down until the message “Programmed path reached (BCO)” is displayed in the message window. The robot stops.

**⚠ CAUTION**

The BCO run is executed as a LIN or PTP motion from the actual position to the target position. The velocity is automatically reduced. The path of the motion cannot be predicted reliably. Observe the motion during the BCO run so that the robot can be stopped in time if a collision becomes imminent.

4. Press Start key and hold it down.

The program is executed with or without stops, depending on the program run mode.

To stop a program that has been started manually, release the Start key.

## 8.8 Starting a program forwards (automatic)

### Precondition

- A program is selected.
- Operating mode Automatic (not Automatic External)

### Procedure

1. Select the program run mode **Go**.

2. Switch on the drives.

3. Carry out a BCO run:

Press Start key and hold it down until the message “Programmed path reached (BCO)” is displayed in the message window. The robot stops.

**⚠ CAUTION**

The BCO run is executed as a LIN or PTP motion from the actual position to the target position. The velocity is automatically reduced. The path of the motion cannot be predicted reliably. Observe the motion during the BCO run so that the robot can be stopped in time if a collision becomes imminent.

4. Press the Start key. The program is executed.

To stop a program that has been started in Automatic mode, press the STOP key.

## 8.9 Carrying out a block selection

**Description** A program can be started at any point by means of a block selection.

### Precondition

- A program is selected.
- Operating mode T1 or T2

### Procedure

1. Select the program run mode.
2. Select the motion block at which the program is to be started.
3. Press **Block selection**. The block pointer indicates the motion block.
4. Hold the enabling switch down and wait until the status bar indicates “Drives ready”:



5. Carry out a BCO run: Press the Start key and hold it down until the message “Programmed path reached (BCO)” is displayed in the message window. The robot stops.



**CAUTION** The BCO run is executed as a LIN or PTP motion from the actual position to the target position. The velocity is automatically reduced. The path of the motion cannot be predicted reliably. Observe the motion during the BCO run so that the robot can be stopped in time if a collision becomes imminent.

6. The program can now be started manually or automatically. It is not necessary to carry out a BCO run again.

## 8.10 Resetting a program

<b>Description</b>	In order to restart an interrupted program from the beginning, it must be reset. This returns the program to the initial state.
<b>Precondition</b>	<ul style="list-style-type: none"> <li>■ Program is selected.</li> </ul>
<b>Procedure</b>	<ul style="list-style-type: none"> <li>■ Select the menu sequence <b>Edit &gt; Reset program</b>.</li> </ul>
<b>Alternative procedure</b>	<ul style="list-style-type: none"> <li>■ In the status bar, touch the <b>Robot interpreter</b> status indicator. A window opens. Select <b>Reset program</b>.</li> </ul>

## 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.

<b>Precondition</b>	<ul style="list-style-type: none"> <li>■ Operating mode T1 or T2</li> <li>■ Inputs/outputs for Automatic External are configured.</li> <li>■ The program CELL.SRC is configured.</li> </ul>
<b>Procedure</b>	<ol style="list-style-type: none"> <li>1. Select the program CELL.SRC in the Navigator. (This program is located in the folder “R1”.)</li> <li>2. Set program override to 100%. (This is the recommended setting. A different value can be set if required.)</li> <li>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.</li> </ol>
	<div data-bbox="426 1686 626 1731" data-label="Image"> </div> <p>The BCO run is executed as a LIN or PTP motion from the actual position to the target position. The velocity is automatically reduced. The path of the motion cannot be predicted reliably. Observe the motion during the BCO run so that the robot can be stopped in time if a collision becomes imminent.</p> <ol style="list-style-type: none"> <li>4. Select “Automatic External” mode.</li> <li>5. Start the program from a higher-level controller (PLC).</li> </ol> <p>To stop a program that has been started in Automatic mode, press the STOP key.</p>

## 8.12 Backward motion using the Start backwards key

### 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 from the actual position to the target position. The velocity is automatically reduced. The path of the motion cannot be predicted reliably. Observe the motion during the BCO run so that the robot can be stopped in time if a collision becomes imminent.

**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.

(>>> 6.18 "Configuring backward motion" Page 194)

`DELETE_BACKWARD_BUFFER()` can be used to prevent backward motion for specific motions.

(>>> 11.16.1 "DELETE\_BACKWARD\_BUFFER()" Page 457)

## 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.	<p>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.</p> <p>Or set \$ADVANCE to "1". This does not always prevent the error message, but it reduces the probability. Approximate positioning is still possible.</p>

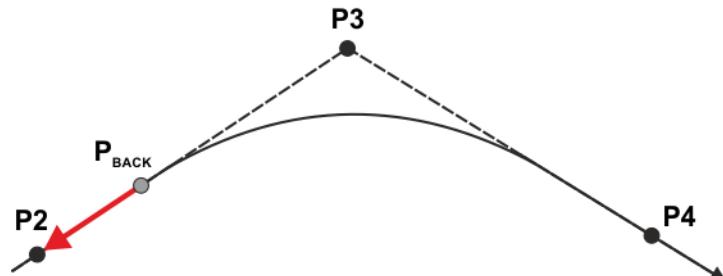
### 8.12.2.2 Approximate positioning response

**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:**

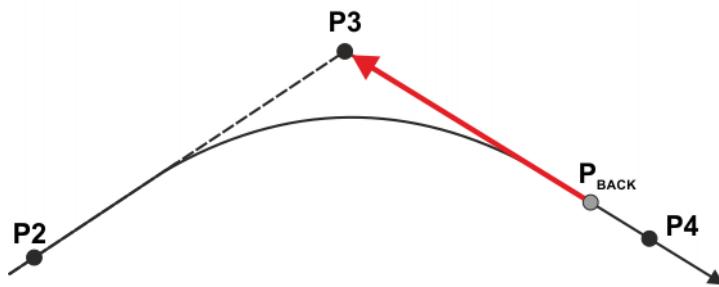
The Start backwards key is pressed while the robot is on the path, but not in an approximate positioning range. The robot now moves backwards on the path to the end point of the previous motion.

$P_{BACK}$  = position of the robot at the moment at which the Start backwards key is pressed



**Fig. 8-14: 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.

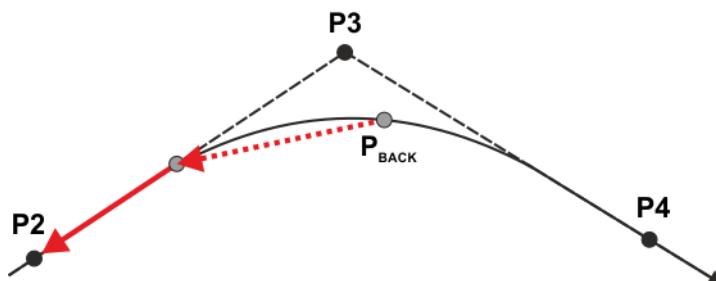


**Fig. 8-15: Case 2: Backward start outside an approximate positioning range**

#### 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. 8-16: Backward start in the approximate positioning range**

#### 8.12.2.3 Response in the case of weave motions

##### Description

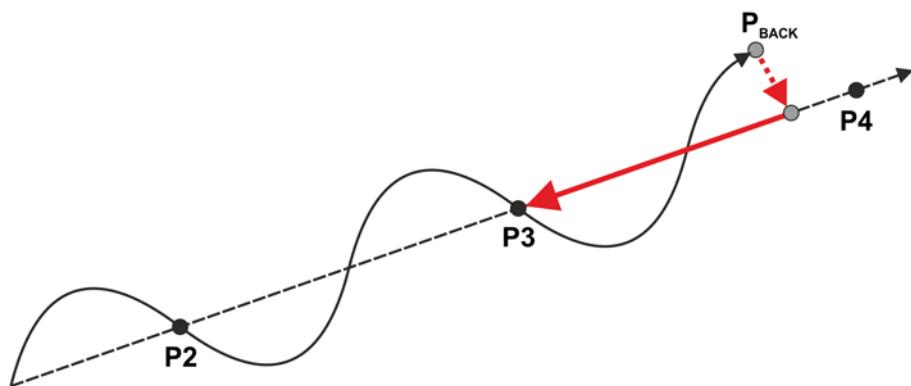
Weaving is not possible during backward motion. If weaving was carried out during forward execution, the backward path will thus differ from the forward path. The robot must therefore perform a BCO run for the backward path after starting backward motion, even though it did not leave the path during 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. 8-17: 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.

<b>Designation</b>	<b>Meaning in V8.3.6 and higher</b>
\$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.
\$VW_RETRACE_AMF	Always supplies the value FALSE. This variable only still exists for reasons of compatibility and 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.



## 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 283)
- **Linear motions (LIN)**  
(>>> 9.3 "Motion type LIN" Page 284)
- **Circular motion (CIRC)**  
(>>> 9.4 "Motion type CIRC" Page 284)
- **Spline motions**

Spline motions have a number of advantages over conventional PTP, LIN and CIRC motions.

(>>> 9.7 "Spline motion type" Page 289)



The start point of a motion is always the end point of the previous motion.

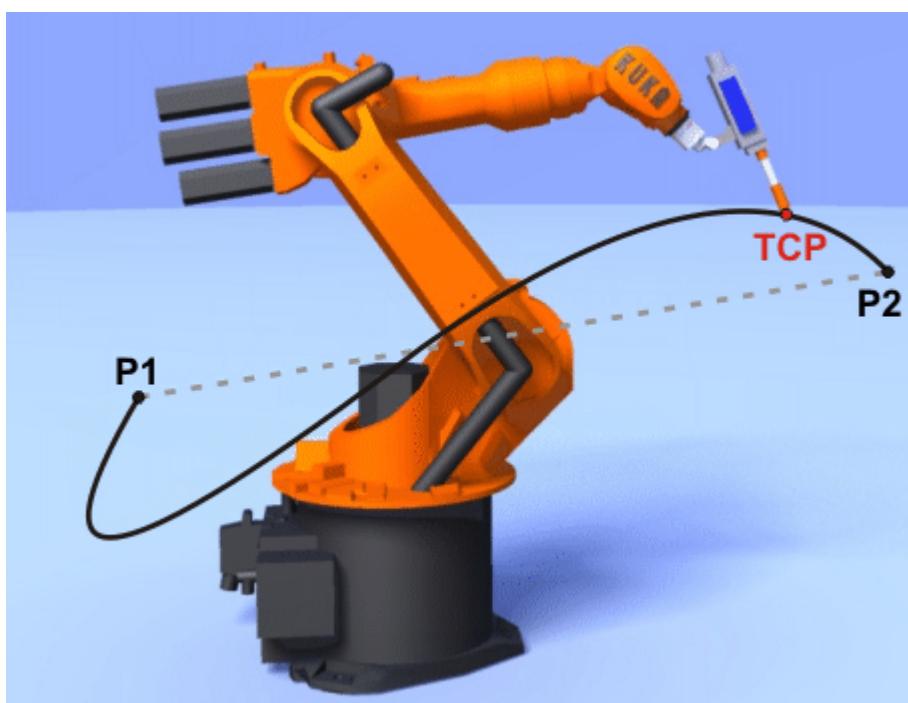
The following motions are known as CP ("Continuous Path") motions.

- LIN, CIRC, CP spline blocks, SLIN, SCIRC

### 9.2 Motion type PTP

The robot guides the TCP along the fastest path to the end point. The fastest path is generally not the shortest path and is thus not a straight line. As the motions of the robot axes are rotational, curved paths can be executed faster than straight paths.

The exact path of the motion cannot be predicted.



**Fig. 9-1: PTP motion**

### 9.3 Motion type LIN

The robot guides the TCP at a defined velocity along a straight path to the end point.

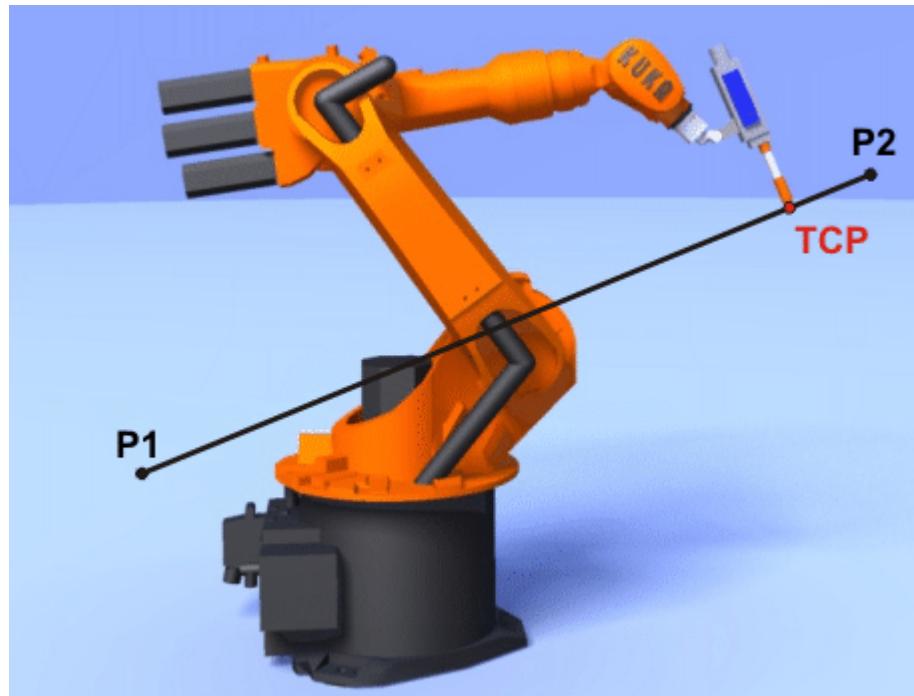


Fig. 9-2: LIN motion

### 9.4 Motion type CIRC

The robot guides the TCP at a defined velocity along a circular path to the end point. The circular path is defined by a start point, auxiliary point and end point.

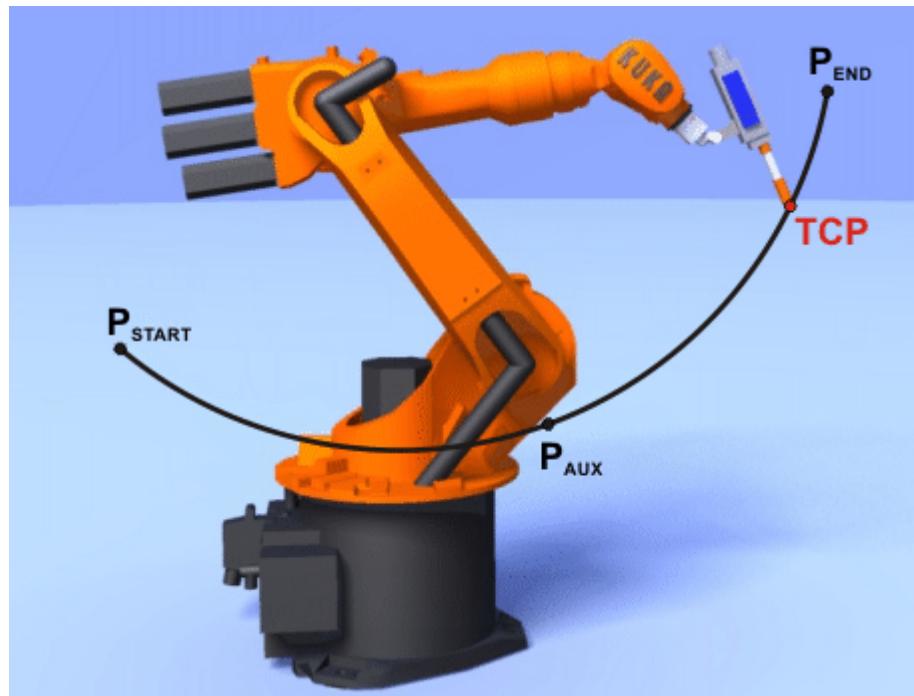


Fig. 9-3: CIRC motion

## 9.5 Approximate positioning

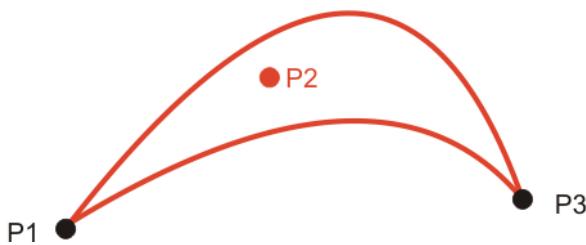
Approximate positioning means that the motion does not stop exactly at the programmed point. Approximate positioning is an option that can be selected during motion programming.

Approximate positioning is not possible if the motion instruction is followed by an instruction that triggers an advance run stop.

### Approximate positioning with a PTP motion

The TCP leaves the path that would lead directly to the end point and moves along a faster path. During programming of the motion, the maximum distance from the end point at which the TCP may deviate from its original path is defined.

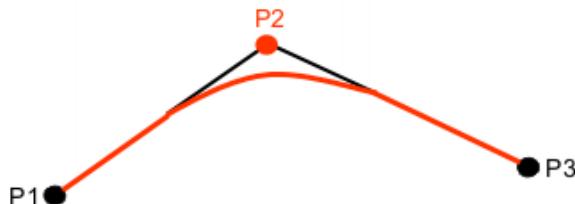
The path of an approximated PTP motion cannot be predicted. It is also not possible to predict on which side of the approximated point the path will run.



**Fig. 9-4: PTP motion, P2 is approximated**

### Approximate positioning with a LIN motion

The TCP leaves the path that would lead directly to the end point and moves along a shorter path. During programming of the motion, the maximum distance from the end point at which the TCP may deviate from its original path is defined.



**Fig. 9-5: LIN motion, P2 is approximated**

### Approximate positioning with a CIRC motion

The TCP leaves the path that would lead directly to the end point and moves along a shorter path. During programming of the motion, the maximum distance from the end point at which the TCP may deviate from its original path is defined.

The motion passes exactly through the auxiliary point.

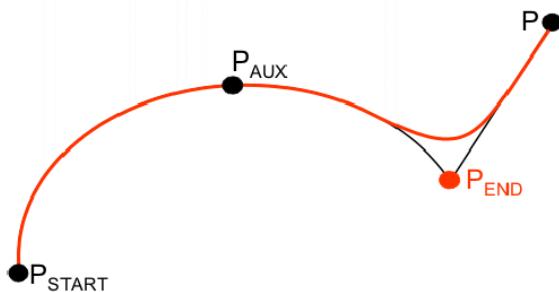


Fig. 9-6: CIRC motion,  $P_{END}$  is approximated

## 9.6 Orientation control LIN, CIRC

### Description

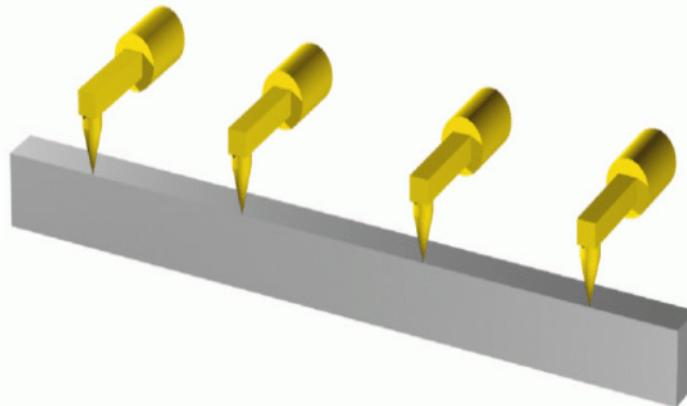
The orientation of the TCP can be different at the start point and end point of a motion. There are several different types of transition from the start orientation to the end orientation. A type must be selected when a CP motion is programmed.

The orientation control for LIN and CIRC motions is defined as follows:

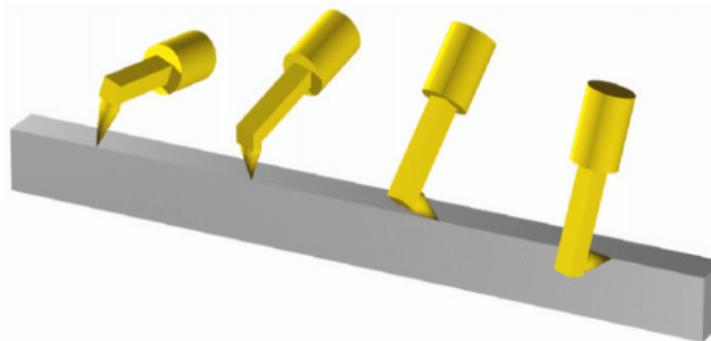
- In the option window **Motion parameters**  
(>>> 10.3.8 "Option window "Motion parameters" (LIN, CIRC, PTP)"  
Page 317)
- Or via the system variable \$ORI\_TYPE

### LIN motion

	Orientation control	Description
	<ul style="list-style-type: none"> <li>■ Option window: <b>Constant orientation</b></li> <li>■ \$ORI_TYPE = #CONSTANT</li> </ul>	<p>The orientation of the TCP remains constant during the motion.</p> <p>The programmed orientation is disregarded for the end point and that of the start point is retained.</p>
	<ul style="list-style-type: none"> <li>■ Option window: <b>Standard</b></li> <li>■ \$ORI_TYPE = #VAR</li> </ul>	<p>The orientation of the TCP changes continuously during the motion.</p> <p><b>Note:</b> If, with <b>Standard</b>, the robot passes through a wrist axis singularity, use <b>Wrist PTP</b> instead.</p>
	<ul style="list-style-type: none"> <li>■ Option window: <b>Wrist PTP</b></li> <li>■ \$ORI_TYPE = #JOINT</li> </ul>	<p>The orientation of the TCP changes continuously during the motion. This is done by linear transformation (axis-specific motion) of the wrist axis angles.</p> <p><b>Note:</b> Use <b>Wrist PTP</b> if, with <b>Standard</b>, the robot passes through a wrist axis singularity.</p> <p>The orientation of the TCP changes continuously during the motion, but not uniformly. <b>Wrist PTP</b> is thus not suitable if a specific orientation must be maintained exactly, e.g. in the case of laser welding.</p>



**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

**i** \$CIRC\_TYPE is meaningless if \$ORI\_TYPE = #JOINT.

(>>> 9.6.1 "Combinations of \$ORI\_TYPE and \$CIRC\_TYPE" Page 287)

#### 9.6.1 Combinations of \$ORI\_TYPE and \$CIRC\_TYPE

**\$ORI\_TYPE = #CONSTANT, \$CIRC\_TYPE = #PATH:**

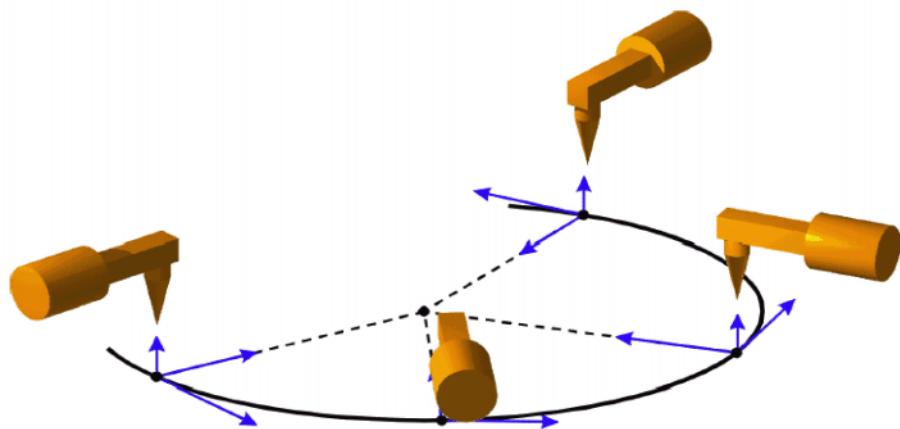


Fig. 9-9: Constant orientation, path-related

`$ORI_TYPE = #VAR, $CIRC_TYPE = #PATH:`

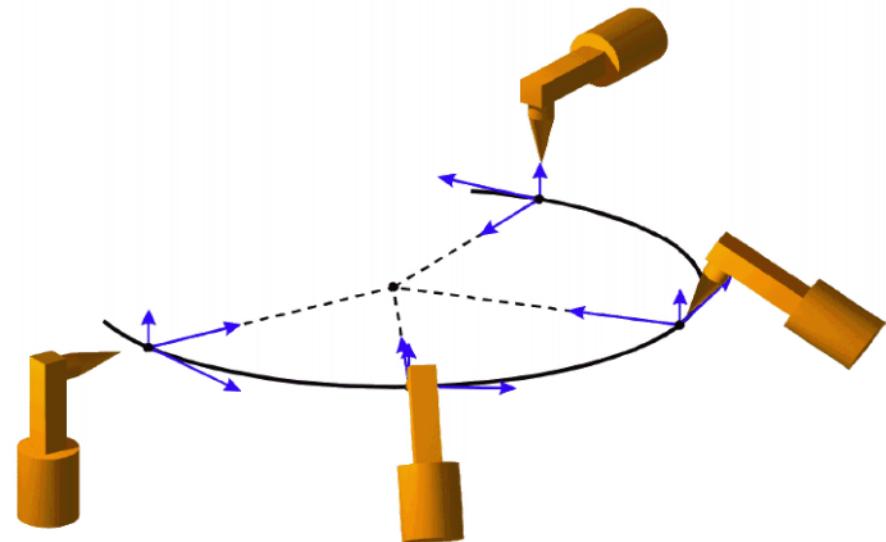


Fig. 9-10: Variable orientation, path-related

`$ORI_TYPE = #CONSTANT, $CIRC_TYPE = #BASE:`

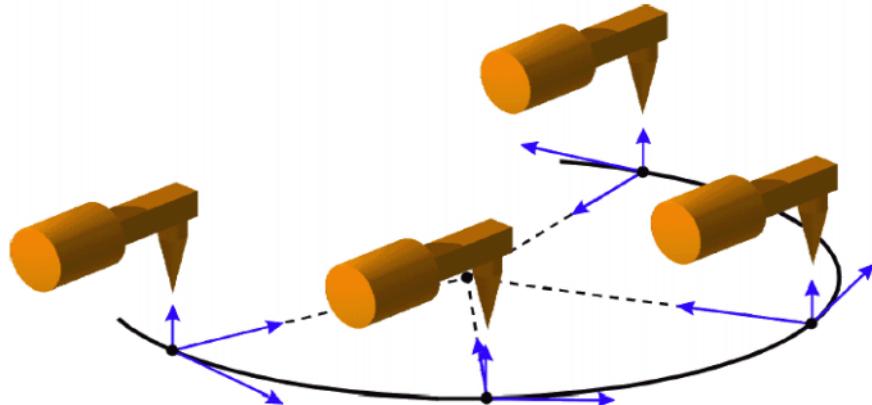
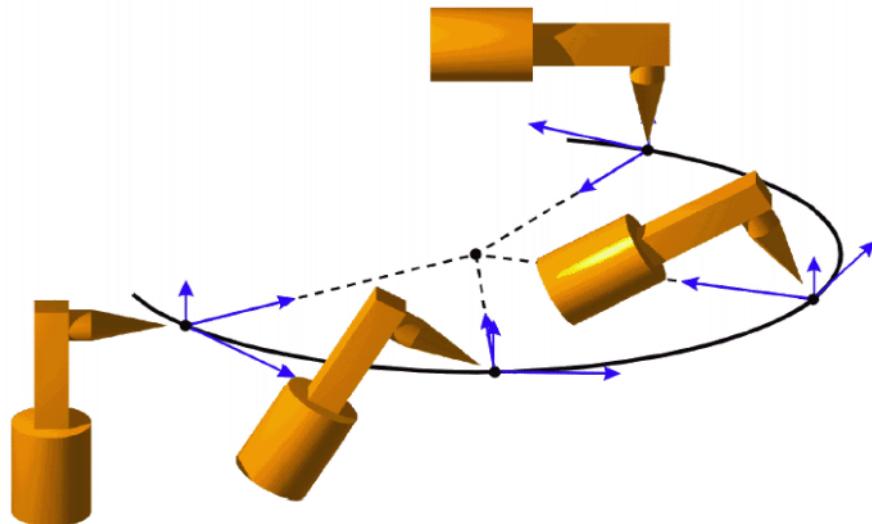


Fig. 9-11: Constant orientation, base-related

**\$ORI\_TYPE = #VAR, \$CIRC\_TYPE = #BASE:**



**Fig. 9-12: Variable orientation, base-related**

### 9.7 Spline motion type

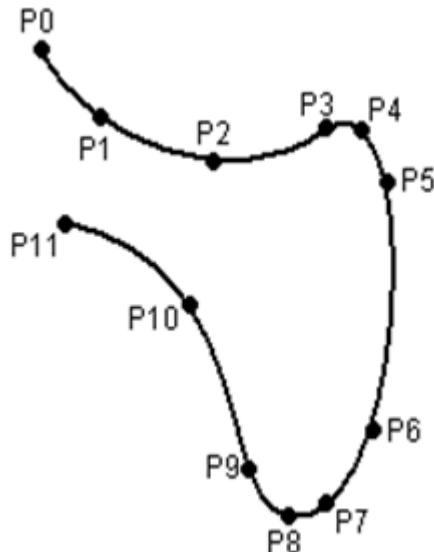
Spline is a motion type that is particularly suitable for complex, curved paths. Such paths can also be generated using approximated LIN and CIRC motions, but splines have advantages, however.

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.

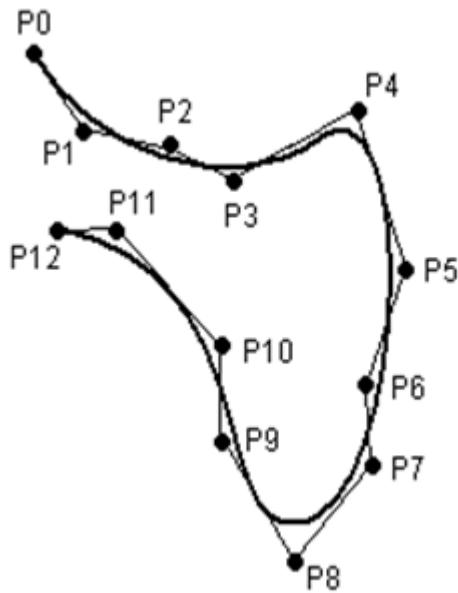
The motions that may be included in a spline block are called spline segments. They are taught separately.

- 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 291)  
Furthermore, special constant velocity ranges can be defined in CP spline blocks.
- The path always remains the same, irrespective of the override setting, velocity or acceleration.
- Circles and tight radii are executed with great precision.

**Disadvantages of LIN/CIRC****Fig. 9-14: Curved path with approximated LIN motions**

- The path is defined by means of approximated points that are not located on the path. The approximate positioning ranges are difficult to predict. Generating the desired path is complicated and time-consuming.

- In many cases, the velocity may be reduced in a manner that is difficult to predict, e.g. in the approximate positioning ranges and near points that are situated close together.
- The path changes if approximate positioning is not possible, e.g. for time reasons.
- The path changes in accordance with the override setting, velocity or acceleration.

### 9.7.1 Velocity profile for spline motions

The path always remains the same, irrespective of the override setting, velocity or acceleration.

The robot controller already takes the physical limits of the robot into consideration during planning. The robot moves as fast as possible within the constraints of the programmed velocity, i.e. as fast as its physical limits will allow. This is an advantage over conventional LIN and CIRC motions for which the physical limits are not taken into consideration during planning. It is only during motion execution that these limits have any effect and can cause stops to be triggered.

#### **Reduction of the velocity**

Prime examples of cases in which the velocity has to fall below the programmed velocity include:

- Tight corners
- Major reorientation
- Large motions of the external axes
- Motion in the vicinity of singularities

Reduction of the velocity due to major reorientation can be avoided with spline segments by selecting the orientation control option **Ignore orientation**.

Reduction of the velocity due to major external axis motions can be avoided for spline segments with \$EX\_AX\_IGNORE.

#### **Reduction of the velocity to 0**

This is the case for:

- Successive points with the same coordinates.
- Successive SLIN and/or SCIRC segments. Cause: inconstant velocity direction.

In the case of SLIN-SCIRC transitions, the velocity is also reduced to 0 if the straight line is a tangent of the circle, as the circle, unlike the straight line, is curved.

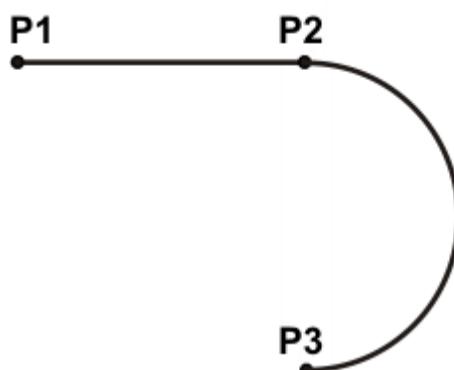
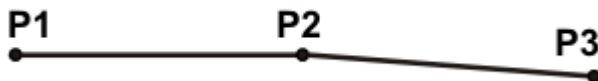


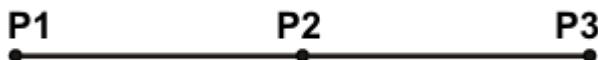
Fig. 9-15: Exact positioning at P2



**Fig. 9-16: Exact positioning at P2**

Exceptions:

- In the case of successive SLIN segments that result in a straight line and in which the orientations change uniformly, the velocity is not reduced.



**Fig. 9-17: P2 is executed without exact positioning.**

- In the case of a SCIRC-SCIRC transition, the velocity is not reduced if both circles have the same center point and the same radius and if the orientations change uniformly. (This is difficult to teach, so calculate and program points.)



Circles with the same center point and the same radius are sometimes programmed to obtain circles  $\geq 360^\circ$ . A simpler method is to program a circular angle.

### 9.7.2 Block selection with spline motions

#### 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. This is indicated by means of a message that must be acknowledged.

- PTP spline block:

The BCO run is executed as a conventional PTP motion. This is not indicated by a message.

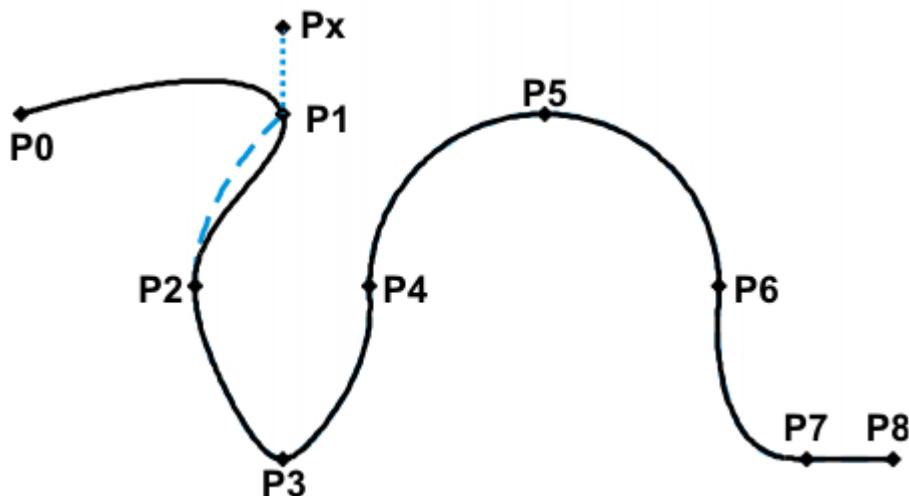
Following a block selection, the path is generally the same as if the spline were to be executed during normal program execution.

Exceptions are possible if the spline has never yet been executed prior to the block selection and the block selection is made here 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**



**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
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 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.

In the case of a block selection to an individual SCIRC motion, the circular angle is never taken into consideration.

### 9.7.3 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:

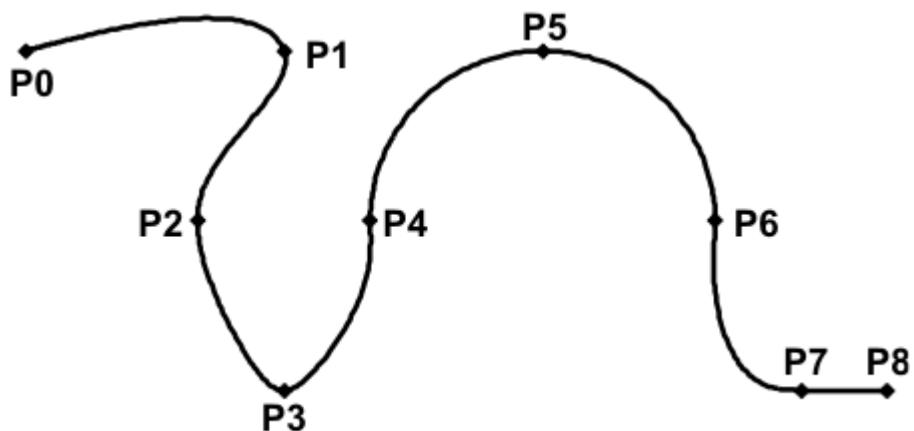
If an SPL segment is changed into an SLIN segment or vice versa, the path changes in the previous segment and the next segment.

**Example 1****Original path:**

```

PTP P0
SPLINE
SPL P1
SPL P2
SPL P3
SPL P4
SCIRC P5, P6
SPL P7
SLIN P8
ENDSPLINE

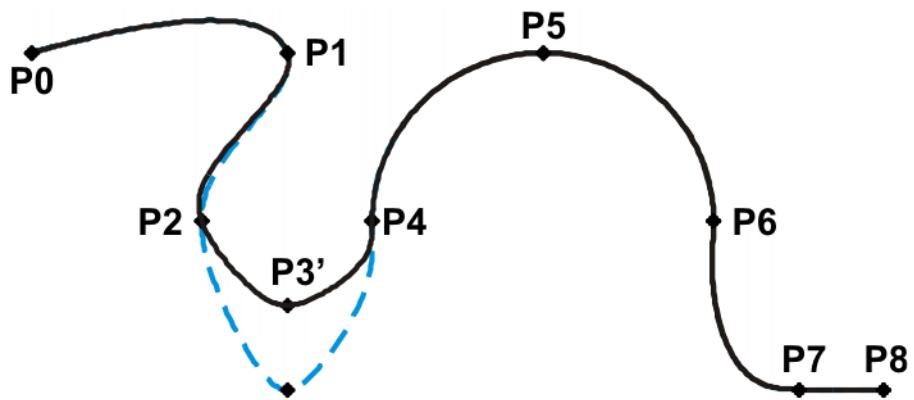
```



**Fig. 9-19: 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. 9-20: Point has been offset**

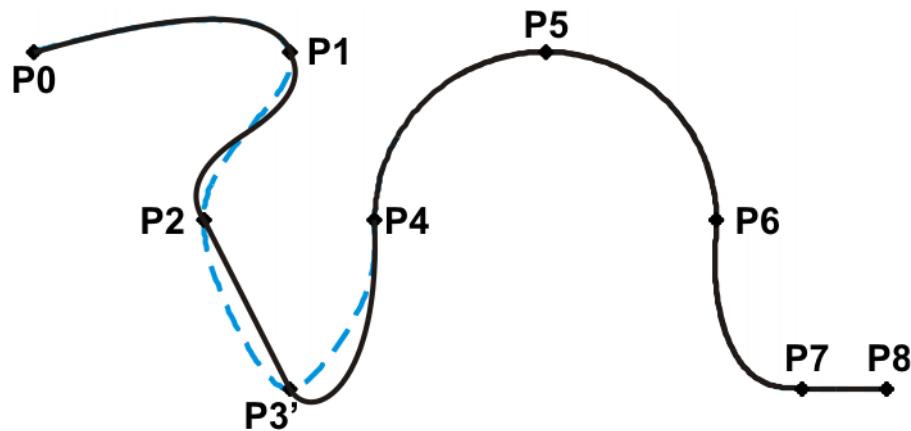
**The type of a segment is changed relative to the original path:**

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. 9-21: Segment type has been changed**

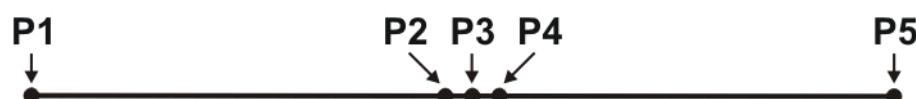
### Example 2

#### 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. 9-22: Original path**

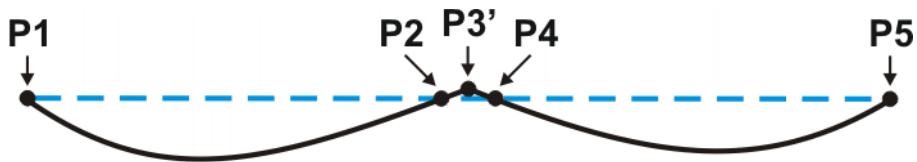
#### A point is offset relative to the original path:

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. 9-23: Point has been offset**

Remedy:

- Distribute the points more evenly
- Program straight lines (except very short ones) as SLIN segments

#### 9.7.4 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.

#### 9.7.5 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.

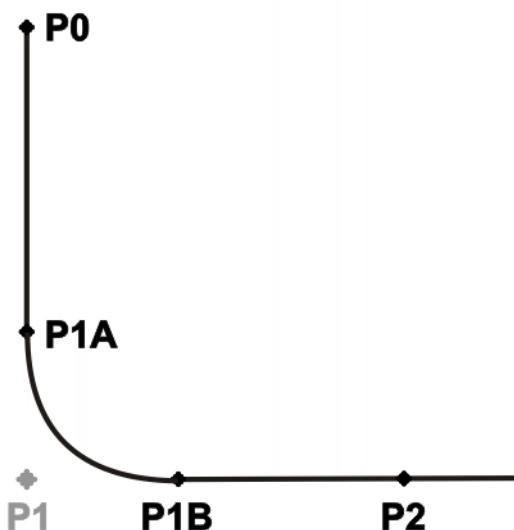
The following approximated motion is to be reproduced:

```
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. 9-24: 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:

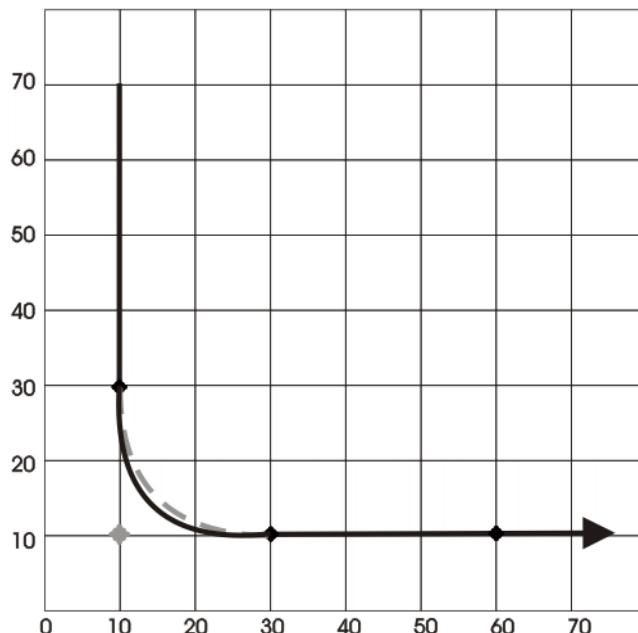
```
$APO.CDIS=20
$VEL.CP=0.5
```

```
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
```

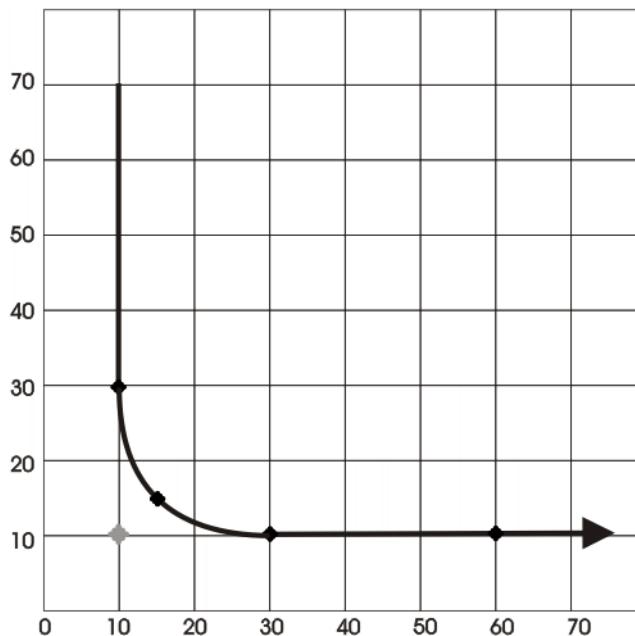
The start of the approximate positioning arc has been calculated from the approximate positioning criterion.



**Fig. 9-25: 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
```

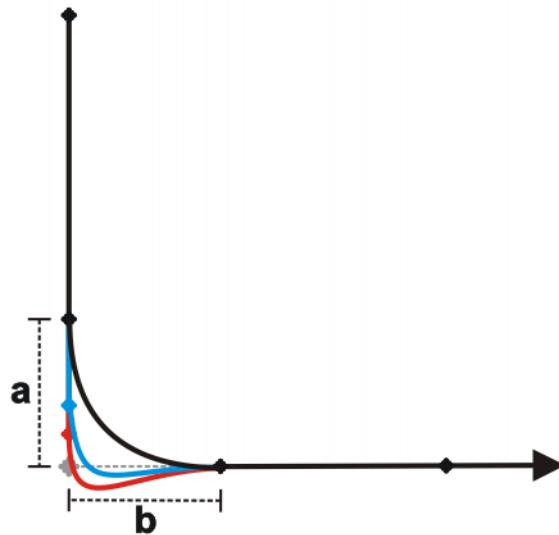


**Fig. 9-26: Example: Approximated motion - spline motion 2**

With the additional point, the path now corresponds to the approximate positioning arc.

#### 9.7.5.1 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. 9-27: SLIN-SPL-SLIN**

The path remains inside if the following conditions are met:

- The extensions of the two SLIN segments intersect.

- $2/3 \leq a/b \leq 3/2$   
 $a$  = 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
<ul style="list-style-type: none"> <li>■ Option window: <b>Constant orientation</b></li> <li>■ \$ORI_TYPE = #CONSTANT</li> </ul>	<p>The orientation of the TCP remains constant during the motion.</p> <p>The orientation of the start point is retained. The programmed orientation of the end point is not taken into consideration.</p>
<ul style="list-style-type: none"> <li>■ Option window: <b>Default</b></li> <li>■ \$ORI_TYPE = #VAR</li> </ul>	<p>The orientation of the TCP changes continuously during the motion. At the end point, the TCP has the programmed orientation.</p>
<ul style="list-style-type: none"> <li>■ Option window: <b>Wrist PTP</b></li> <li>■ \$ORI_TYPE = #JOINT</li> </ul>	<p>The orientation of the TCP changes continuously during the motion. This is done by linear transformation (axis-specific motion) of the wrist axis angles.</p> <p><b>Note:</b> Use <b>Wrist PTP</b> if, with <b>Default</b>, the robot passes through a wrist axis singularity.</p> <p>The orientation of the TCP changes continuously during the motion, but not uniformly. <b>Wrist PTP</b> is thus not suitable if a specific orientation must be maintained exactly, e.g. in the case of laser welding.</p>
<ul style="list-style-type: none"> <li>■ Option window: <b>Ignore orientation</b></li> <li>■ \$ORI_TYPE = #IGNORE</li> </ul>	<p>This option is only available for CP spline segments (not for the spline block or for individual spline motions).</p> <p>This option is used if no specific orientation is required at a specific point.</p> <p>(&gt;&gt;&gt; "#IGNORE" Page 301)</p>

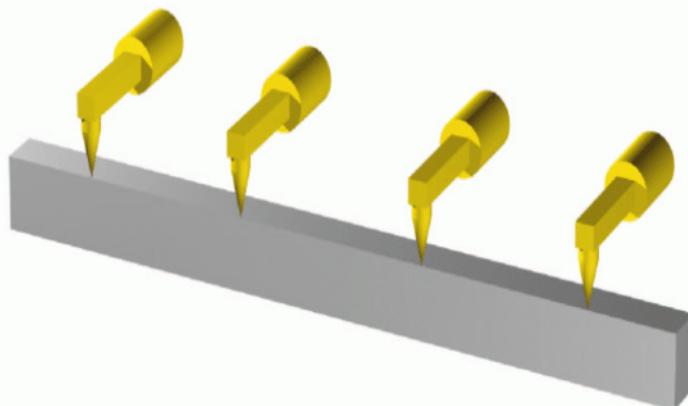
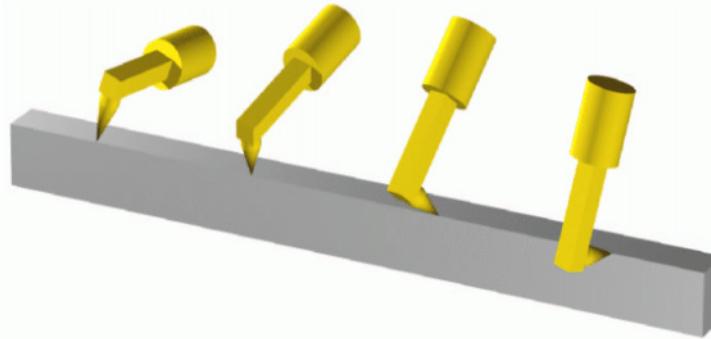


Fig. 9-28: Constant orientation



**Fig. 9-29: 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 302)

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 302)

#### \$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
<ul style="list-style-type: none"> <li>Option window: <b>base-related</b></li> <li>\$CIRC_TYPE = #BASE</li> </ul>	Base-related orientation control during the circular motion
<ul style="list-style-type: none"> <li>Option window: <b>path-oriented</b></li> <li>\$CIRC_TYPE = #PATH</li> </ul>	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 287)

### 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
<code>BehaviorAUX</code>	<p>Data type: ENUM</p> <ul style="list-style-type: none"> <li>■ <b>#INTERPOLATE</b>: At the auxiliary point, the TCP adopts the programmed orientation.</li> <li>■ <b>#IGNORE</b>: 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.</li> <li>■ <b>#CONSIDER</b> (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.</li> </ul>
<code>BehaviorEND</code>	<p>Data type: ENUM</p> <ul style="list-style-type: none"> <li>■ <b>#INTERPOLATE</b>: 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.)</li> <li>■ <b>#EXTRAPOLATE</b>: 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.)</li> </ul>

### 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°.

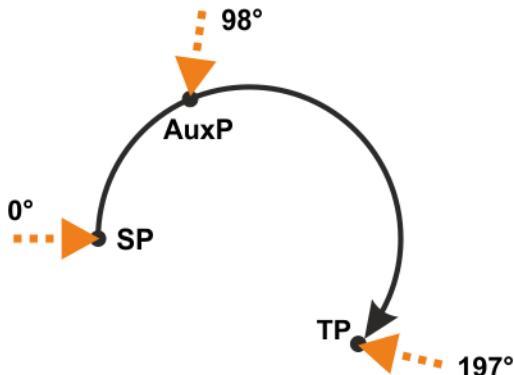


Fig. 9-30: #INTERPOLATE

- |             |                 |
|-------------|-----------------|
| <b>SP</b>   | Start point     |
| <b>AuxP</b> | Auxiliary point |
| <b>TP</b>   | End point       |

#### #IGNORE

The short re-orientation through 163° is used. The programmed orientation of the auxiliary point is disregarded.

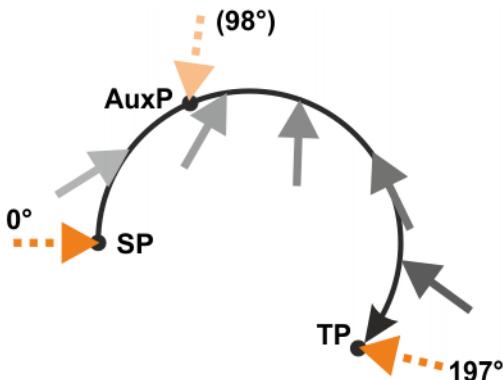


Fig. 9-31: #IGNORE

#### #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.

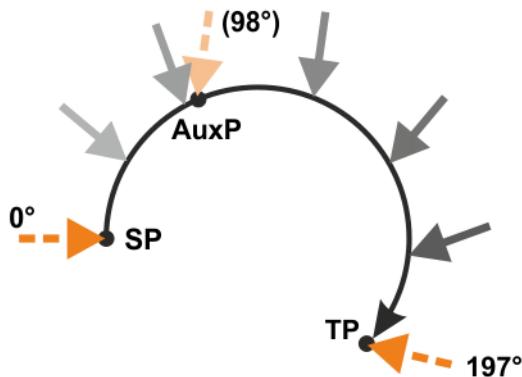


Fig. 9-32: #CONSIDER

#### 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.

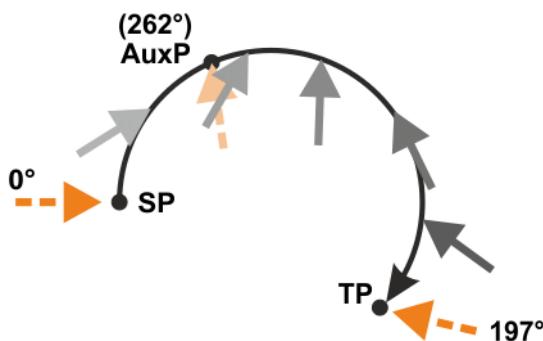


Fig. 9-33: #CONSIDER, additional example

#### 9.8.2.2 SCIRC: Orientation behavior – example: end point

<b>Description</b>	<ul style="list-style-type: none"> <li>■ The dotted orange arrows show the programmed orientation.</li> <li>■ The gray arrows show the actual orientation where this differs from the programmed orientation.</li> </ul>
<b>#INTERPOLATE</b>	At <b>TP</b> , which is situated before <b>TP_CA</b> , the programmed orientation has not yet been reached. The programmed orientation is accepted at <b>TP_CA</b> .

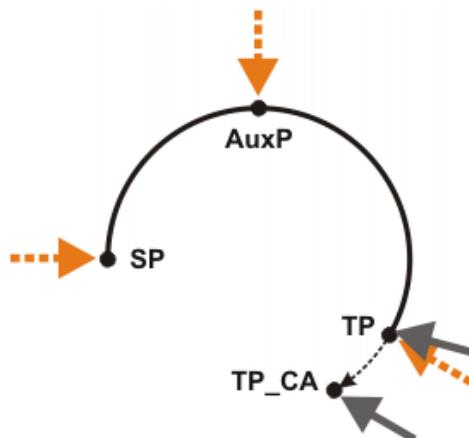


Fig. 9-34: #INTERPOLATE

- 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.

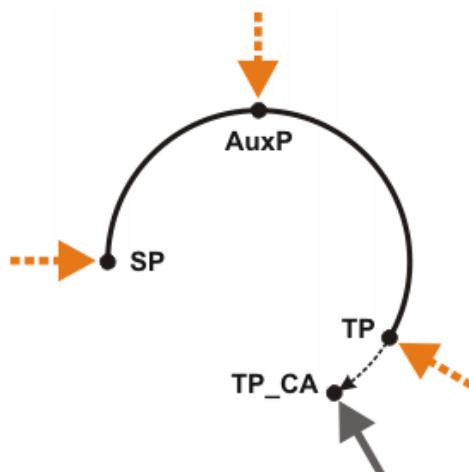


Fig. 9-35: #EXTRAPOLATE

#### 9.9 Circular angle

A circular angle can be programmed for most circular motions.



Information about whether this is possible for a specific circular motion can be found in the description of the motion in the programming section of this documentation.

The circular angle specifies the overall angle of the motion. This makes it possible to extend the motion beyond the programmed end point or to shorten it. The actual end point thus no longer corresponds to the programmed end point.

Unit: degrees. Circular angles greater than  $+360^\circ$  and less than  $-360^\circ$  can be programmed.

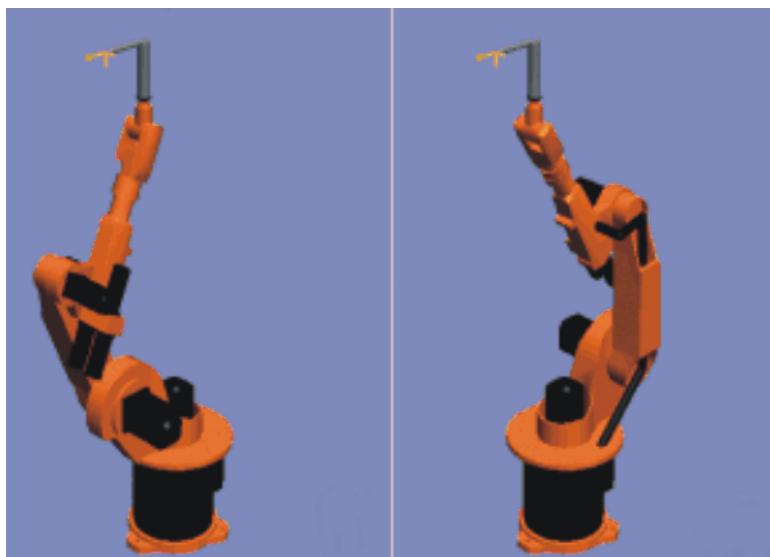
The preceding sign determines the direction in which the circular path is executed:

- 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. 9-36: Example: Same TCP position, different axis position**

Status (S) and Turn (T) are integral parts of the data types POS and E6POS:

```
STRUC POS REAL X, Y, Z, A, B, C, INT S, T
```

```
STRUC E6POS REAL X, Y, Z, A, B, C, E1, E2, E3, E4, E5, E6, INT S, T
```

### KRL program

The robot controller only takes the programmed Status and Turn values into consideration for PTP motions. They are ignored for CP motions.

The first motion instruction in a KRL program must therefore be one of the following instructions so that an unambiguous starting position is defined for the robot:

- A complete PTP instruction of type POS or E6POS
- Or a complete PTP instruction of type AXIS or E6AXIS

“Complete” means that all components of the end point must be specified. The default HOME position is always a complete PTP instruction.

Status and Turn can be omitted in the subsequent instructions:

- The robot controller retains the previous Status value.
- The Turn value is determined by the path in CP motions. In the case of PTP motions, the robot controller selects the Turn value that results in the shortest possible path.

### 9.10.1 Status

The Status consists of a sequence of bits which, together with the position of the TCP, defines the axis position of the robot.

**Bit 0**

Bit 0 specifies the position of the intersection of the wrist axes (A4, A5, A6).

Position	Value
Overhead area If the x-value of the intersection of the wrist axes, relative to the A1 coordinate system, is negative, the robot is in the overhead area.	Bit 0 = 1
Basic area If the x-value of the intersection of the wrist axes, relative to the A1 coordinate system, is positive, the robot is in the basic area.	Bit 0 = 0

The A1 coordinate system is identical to the \$ROBROOT coordinate system if axis 1 is at 0°. For values not equal to 0°, it moves with axis 1.

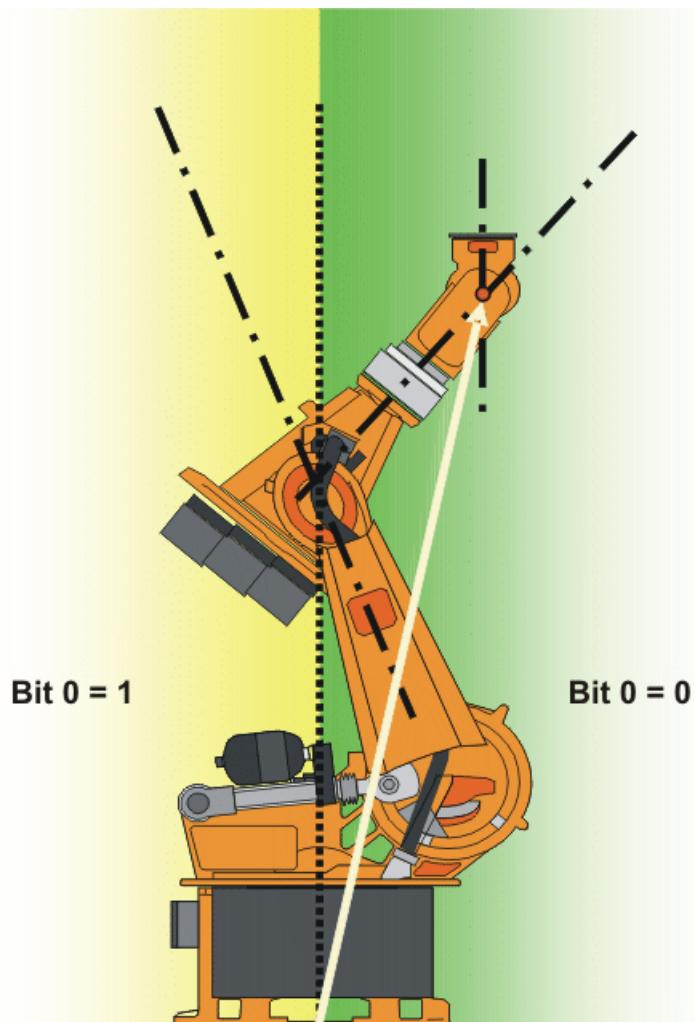
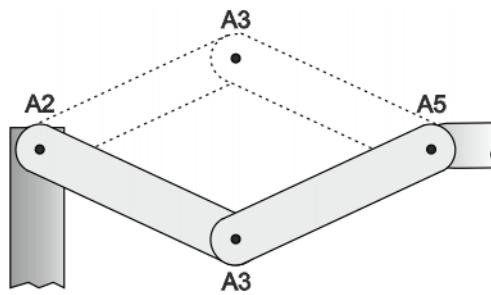


Fig. 9-37: Example: The intersection of the wrist axes (red dot) is in the basic area.

**Bit 1**

Bit 1 specifies the position of axis A3.



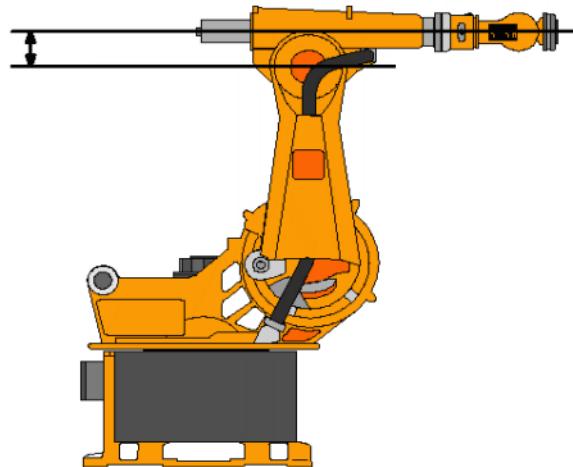
**Fig. 9-38: Bit 1 – position of A3**

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 $\geq 0^\circ$	Bit 1 = 1
A3 $< 0^\circ$	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.



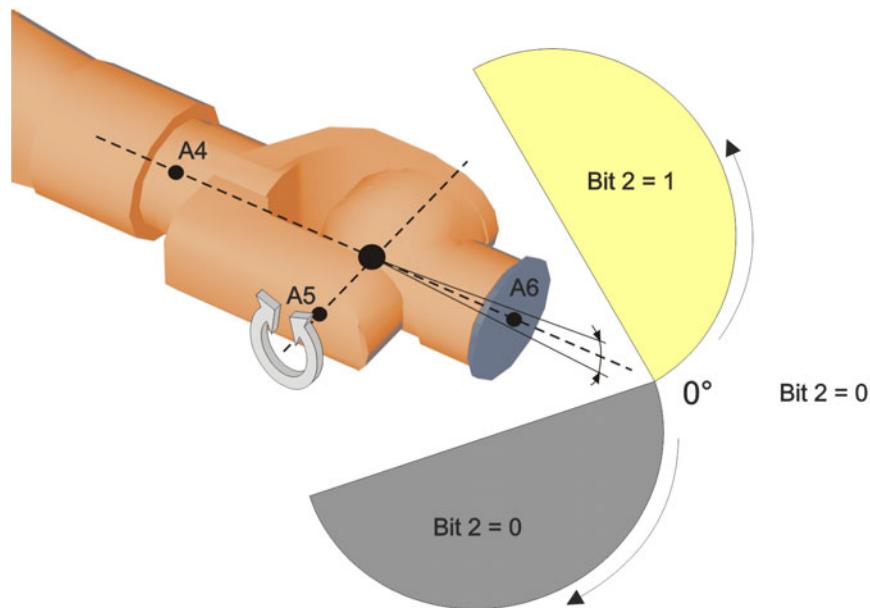
**Fig. 9-39: Offset between A3 and A4: example KR 30**

## 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. 9-40: 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  $+180^\circ$  or less than  $-180^\circ$  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  $\geq 0^\circ$

Bit = 1: Angle  $< 0^\circ$

Value	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	$A6 \geq 0^\circ$	$A5 \geq 0^\circ$	$A4 \geq 0^\circ$	$A3 \geq 0^\circ$	$A2 \geq 0^\circ$	$A1 \geq 0^\circ$
1	$A6 < 0^\circ$	$A5 < 0^\circ$	$A4 < 0^\circ$	$A3 < 0^\circ$	$A2 < 0^\circ$	$A1 < 0^\circ$

##### 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  $\pm 0.01812^\circ$ .

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 for user group "User" (inline forms)

### 10.1 Instructions for programming

**NOTICE**

When programming motions, it must be ensured that the energy supply system is not wound up or damaged during program execution.

**NOTICE**

In the case of programs with the following axis motions or positions, the film of lubricant on the gear units of the axes may break down:

- Motions <3°
- Oscillating motions
- Areas of gear units permanently facing upwards

It must be ensured that the gear units have a sufficient supply of oil. For this, in the case of oscillating motions or short motions (<3°), programming must be carried out in such a way that the affected axes regularly move more than 40° (e.g. once per cycle).

In the case of areas of gear units permanently facing upwards, sufficient oil supply must be achieved by programming re-orientations of the 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.

### 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
- No special characters are permissible, with the exception of \$.
- The first character must not be a number.

The restrictions do not apply to output names.

Other restrictions may apply in the case of inline forms in technology packages.

### 10.3 Programming PTP, LIN and CIRC motions

#### 10.3.1 Programming a PTP motion

**Precondition**

- A program is selected.
- Operating mode T1

**Procedure**

1. Move the TCP to the position that is to be taught as the end point.
2. Position the cursor in the line after which the motion instruction is to be inserted.
3. Select the menu sequence **Commands > Motion > PTP**.
4. Set the parameters in the inline form.

(>>> 10.3.2 "Inline form "PTP"" Page 314)

5. Save instruction with **Cmd OK**.

### 10.3.2 Inline form "PTP"

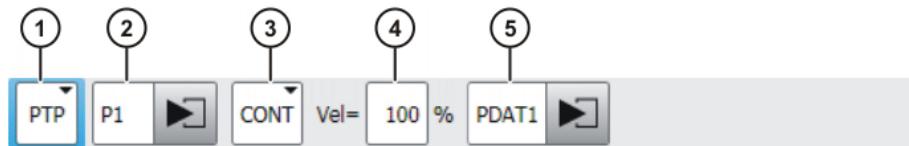


Fig. 10-1: Inline form for PTP motions

Item	Description
1	Motion type <b>PTP</b>
2	Name of the end point The system automatically generates a name. The name can be overwritten. (>>> 10.2 "Names in inline forms" Page 313) Touch the arrow to edit the point data. The corresponding option window is opened. (>>> 10.3.7 "Option window "Frames"" Page 316)
3	■ <b>CONT</b> : End point is approximated. ■ <b>[blank]</b> : The motion stops exactly at the end point.
4	Velocity ■ <b>1 ... 100%</b>
5	Name for the motion data set The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the point data. The corresponding option window is opened. (>>> 10.3.8 "Option window "Motion parameters" (LIN, CIRC, PTP)" Page 317)

### 10.3.3 Programming a LIN motion

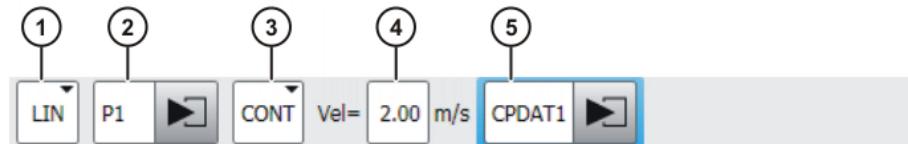
#### Precondition

- A program is selected.
- Operating mode T1

#### Procedure

1. Move the TCP to the position that is to be taught as the end point.
2. Position the cursor in the line after which the motion instruction is to be inserted.
3. Select the menu sequence **Commands > Motion > LIN**.
4. Set the parameters in the inline form.  
(>>> 10.3.4 "Inline form "LIN"" Page 315)
5. Save instruction with **Cmd OK**.

### 10.3.4 Inline form "LIN"



**Fig. 10-2: Inline form for LIN motions**

Item	Description
1	Motion type <b>LIN</b>
2	Name of the end point The system automatically generates a name. The name can be overwritten. (>>> 10.2 "Names in inline forms" Page 313) Touch the arrow to edit the point data. The corresponding option window is opened. (>>> 10.3.7 "Option window "Frames"" Page 316)
3	■ <b>CONT</b> : End point is approximated. ■ <b>[blank]</b> : The motion stops exactly at the end point.
4	Velocity ■ <b>0.001 ... 2 m/s</b>
5	Name for the motion data set The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the point data. The corresponding option window is opened. (>>> 10.3.8 "Option window "Motion parameters" (LIN, CIRC, PTP)" Page 317)

### 10.3.5 Programming a CIRC motion

- Precondition**
- A program is selected.
  - Operating mode T1
- Procedure**
1. Move the TCP to the position that is to be taught as the auxiliary point.
  2. Position the cursor in the line after which the motion instruction is to be inserted.
  3. Select the menu sequence **Commands > Motion > CIRC**.
  4. Set the parameters in the inline form.  
(>>> 10.3.6 "Inline form "CIRC"" Page 316)
  5. Press **Teach Aux**.
  6. Move the TCP to the position that is to be taught as the end point.
  7. Save instruction with **Cmd OK**.

### 10.3.6 Inline form “CIRC”

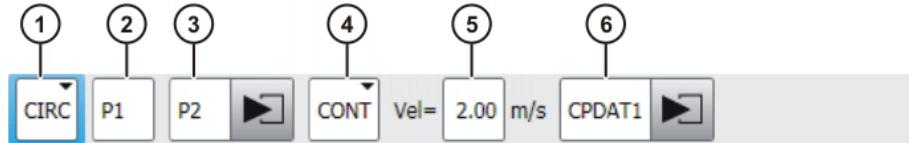


Fig. 10-3: Inline form for CIRC motions

Item	Description
1	Motion type <b>CIRC</b>
2	Name of the auxiliary point The system automatically generates a name. The name can be overwritten. (>>> 10.2 "Names in inline forms" Page 313)
3	Name of the end point The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the point data. The corresponding option window is opened. (>>> 10.3.7 "Option window “Frames”" Page 316)
4	<ul style="list-style-type: none"> <li>■ <b>CONT</b>: End point is approximated.</li> <li>■ <b>[blank]</b>: The motion stops exactly at the end point.</li> </ul>
5	Velocity <ul style="list-style-type: none"> <li>■ <b>0.001 ... 2 m/s</b></li> </ul>
6	Name for the motion data set The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the point data. The corresponding option window is opened. (>>> 10.3.8 "Option window “Motion parameters” (LIN, CIRC, PTP)" Page 317)

### 10.3.7 Option window “Frames”

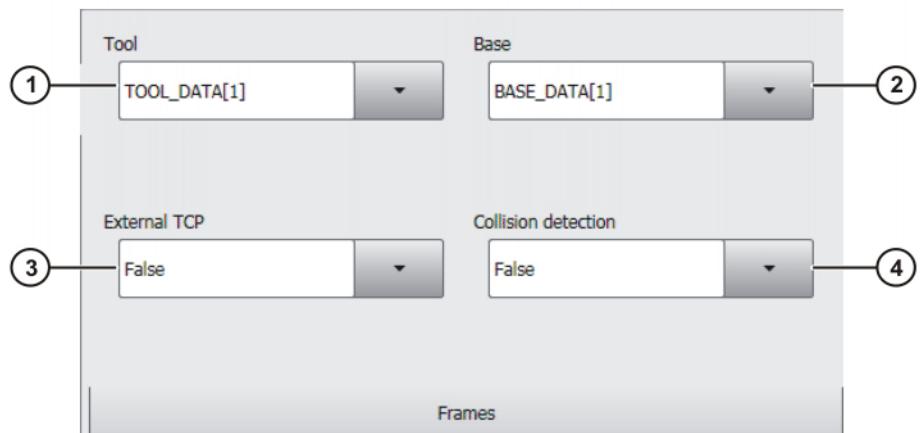


Fig. 10-4: Option window “Frames”

Item	Description
1	Tool selection. If <b>True</b> in the box <b>External TCP</b> : workpiece selection. Range of values: [1] ... [16]
2	Base selection. If <b>True</b> in the box <b>External TCP</b> : fixed tool selection. Range of values: [1] ... [32]
3	Interpolation mode <ul style="list-style-type: none"> <li>■ <b>False</b>: The tool is mounted on the mounting flange.</li> <li>■ <b>True</b>: The tool is a fixed tool.</li> </ul>
4	<ul style="list-style-type: none"> <li>■ <b>True</b>: For this motion, the robot controller calculates the axis torques. These are required for collision detection.</li> <li>■ <b>False</b>: For this motion, the robot controller does not calculate the axis torques. Collision detection is thus not possible for this motion.</li> </ul>

#### 10.3.8 Option window "Motion parameters" (LIN, CIRC, PTP)

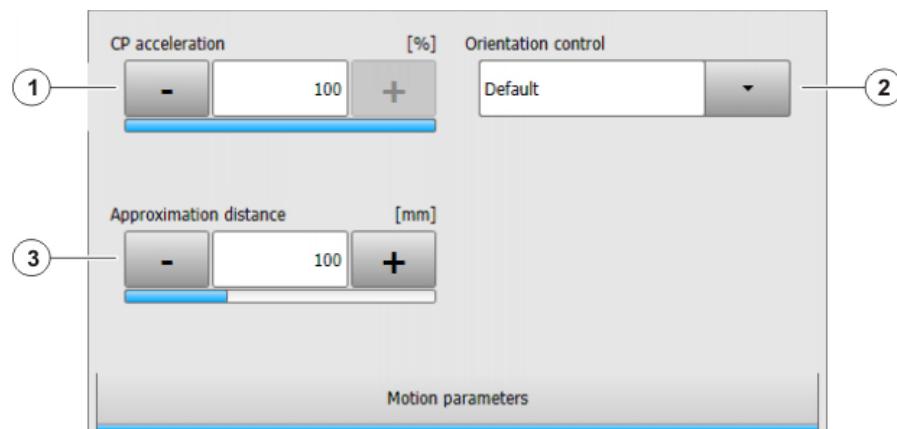


Fig. 10-5: Option window "Motion parameters" (LIN, CIRC, PTP)

Item	Description
1	Acceleration Refers to the maximum value specified in the machine data. The maximum value depends on the robot type and the selected operating mode.

Item	Description
2	<p>This box is only displayed if it is specified in the inline form that the point is to be approximated.</p> <p>Furthest distance before the end point at which approximate positioning can begin</p> <p>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.</p>
3	<p>This box is only displayed for LIN and CIRC motions.</p> <p>Orientation control selection.</p> <ul style="list-style-type: none"> <li>■ <b>Standard</b></li> <li>■ <b>Wrist PTP</b></li> <li>■ <b>Constant orientation</b></li> </ul> <p>(&gt;&gt;&gt; 9.6 "Orientation control LIN, CIRC" Page 286)</p>

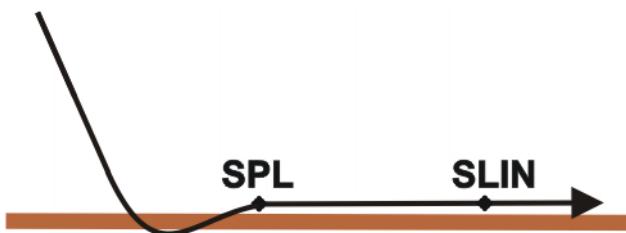
## 10.4 Programming spline motions

### 10.4.1 Programming tips for spline motions

- It is only possible to exploit the advantages of the spline motion type to the full if spline blocks are used.
- Interrupt programs must not contain any spline motions.
- A spline block should cover only one process (e.g. an adhesive seam). More than one process in a spline block leads to a loss of structural clarity within the program and makes changes more difficult.
- Use SLIN and SCIRC segments in cases where the workpiece necessitates straight lines and arcs. (Exception: use SPL segments for very short straight lines.) Otherwise, use SPL segments, particularly if the points are close together.
- Procedure for defining the path:
  - a. First teach or calculate a few characteristic points. Example: points at which the curve changes direction.
  - b. 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.

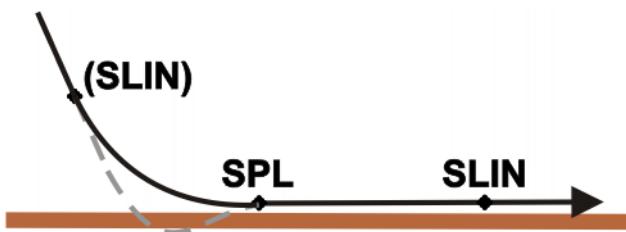
- The jerk can be modified. The jerk is the change in acceleration. Procedure:
  - a. Use the default values initially.
  - b. If vibrations occur at tight corners: reduce values.
 If the velocity drops or the desired velocity cannot be reached: increase values or increase acceleration.
- If the robot executes points which lie on a work surface, a collision with the work surface is possible when approaching the first point.



**Fig. 10-6: Collision with work surface**

In order to avoid a collision, observe the recommendations for the SLIN-SPL-SLIN transition.

(>>> 9.7.5.1 "SLIN-SPL-SLIN transition" Page 299)



**Fig. 10-7: Avoiding a collision with the work surface**

- In the case of PTP spline blocks with multiple SPTP segments, it is possible that the software limit switches may be violated even though the points are within the limits!
- In this case, the points must be re-taught, i.e. they must be moved further away from the software limit switches. Alternatively, the software limit switches can be modified, provided that the required machine protection is still assured.

#### 10.4.2 Programming a spline block

##### Description

A spline block can be used to group together several motions as an overall motion. The motions that may be included in a spline block are called spline segments. They are taught separately.

A spline block is planned and executed by the robot controller as a single motion block.

- A CP spline block may contain SPL, SLIN and SCIRC segments.
- A PTP spline block may contain SPTP segments.

A spline block that contains no segments is not a motion statement. The number of segments in the block is only limited by the memory capacity. Apart from the segments, a spline block may also contain the following elements:

- Inline commands from technology packages that support the spline functionality

- Comments and blank lines

A spline block must not include any other instructions, e.g. variable assignments or logic statements.



The start point of a spline block is the last point before the spline block.

The end point of a spline block is the last point in the spline block.  
A spline block does not trigger an advance run stop.

**Precondition**

- A program is selected.
- Operating mode T1

**Procedure**

1. Position the cursor in the line after which the spline block is to be inserted.
2. Select the menu sequence **Commands > Motion**.
  - Then select **SPLINE block** for a CP spline block.
  - Or select **PTP SPLINE block** for a PTP spline block.
3. Set the parameters in the inline form.
 

(>>> 10.4.2.1 "Inline form for CP spline block" Page 320)

(>>> 10.4.2.2 "Inline form "PTP SPLINE block"" Page 321)
4. Press **Cmd OK**.
5. Press **Open/close fold**. Spline segments can now be inserted into the block.

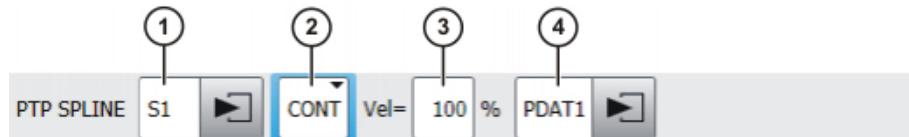
#### 10.4.2.1 Inline form for CP spline block



**Fig. 10-8: Inline form for CP spline block**

Item	Description
1	Name of the spline block. The system automatically generates a name. The name can be overwritten. (>>> 10.2 "Names in inline forms" Page 313) Position the cursor in this box to edit the motion data. The corresponding option window is opened. (>>> 10.4.2.3 "Option window "Frames" (CP and PTP spline block)" Page 321)
2	■ <b>CONT</b> : End point is approximated. ■ <b>[blank]</b> : The motion stops exactly at the end point.
3	Cartesian velocity ■ <b>0.001 ... 2 m/s</b>
4	Name for the motion data set. The system automatically generates a name. The name can be overwritten. Position the cursor in this box to edit the motion data. The corresponding option window is opened. (>>> 10.4.2.4 "Option window "Motion parameters" (CP spline block)" Page 322)

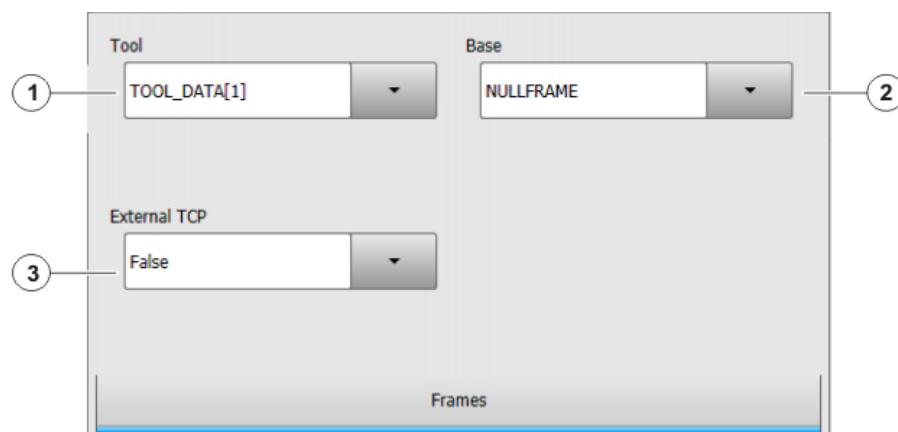
#### 10.4.2.2 Inline form “PTP SPLINE block”



**Fig. 10-9: Inline form “PTP SPLINE block”**

Item	Description
1	Name of the spline block. The system automatically generates a name. The name can be overwritten. (>>> 10.2 "Names in inline forms" Page 313) Position the cursor in this box to edit the motion data. The corresponding option window is opened. (>>> 10.4.2.3 "Option window “Frames” (CP and PTP spline block)" Page 321)
2	■ <b>CONT:</b> End point is approximated. ■ <b>[blank]:</b> The motion stops exactly at the end point.
3	Axis velocity ■ <b>1 ... 100%</b>
4	Name for the motion data set. The system automatically generates a name. The name can be overwritten. Position the cursor in this box to edit the motion data. The corresponding option window is opened. (>>> 10.4.2.5 "Option window “Motion parameters” (PTP spline block)" Page 323)

#### 10.4.2.3 Option window “Frames” (CP and PTP spline block)



**Fig. 10-10: Option window “Frames” (CP and PTP spline block)**

Item	Description
1	Tool selection. Or: If <b>True</b> in the box <b>External TCP</b> : workpiece selection. ■ [1] ... [16]
2	Base selection. Or: If <b>True</b> in the box <b>External TCP</b> : fixed tool selection. ■ [1] ... [32]
3	Interpolation mode ■ <b>False</b> : The tool is mounted on the mounting flange. ■ <b>True</b> : The tool is a fixed tool.

#### 10.4.2.4 Option window “Motion parameters” (CP spline block)

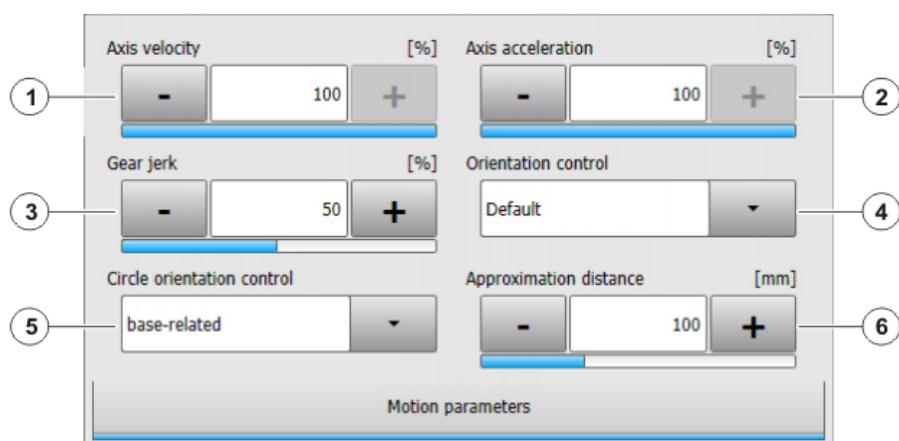
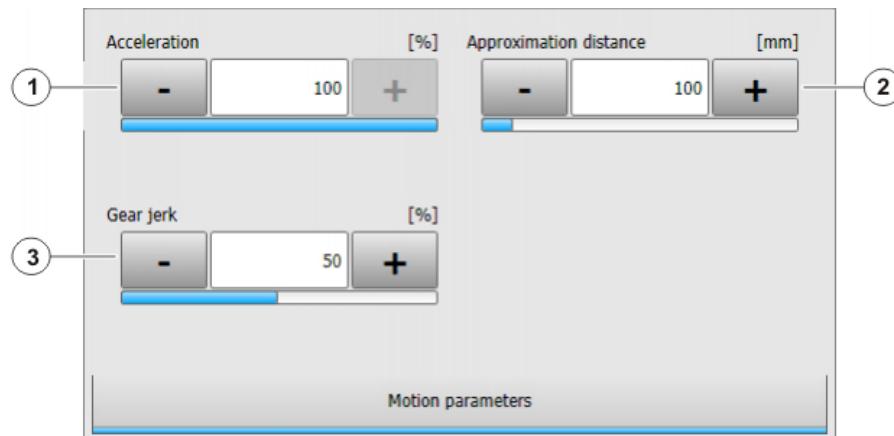


Fig. 10-11: Option window “Motion parameters” (CP spline block)

Item	Description
1	Axis velocity. The value refers to the maximum value specified in the machine data. ■ 1 ... 100%
2	Axis acceleration. The value refers to the maximum value specified in the machine data. ■ 1 ... 100%
3	Gear jerk. The jerk is the change in acceleration. The value refers to the maximum value specified in the machine data. ■ 1 ... 100%
4	Orientation control selection.
5	Orientation control reference system selection. This parameter only affects SCIRC segments (if present).
6	This box is only displayed if <b>CONT</b> was selected in the inline form. Furthest distance before the end point at which approximate positioning can begin. The maximum distance is that of the last segment in the spline. If there is only one segment present, the maximum distance is half the segment length. If a higher value is entered, this is ignored and the maximum value is used.

#### 10.4.2.5 Option window “Motion parameters” (PTP spline block)



**Fig. 10-12: Option window “Motion parameters” (PTP spline block)**

Item	Description
1	Axis acceleration. The value refers to the maximum value specified in the machine data. ■ 1 ... 100%
2	This box is only displayed if <b>CONT</b> was selected in the inline form. Furthest distance before the end point at which approximate positioning can begin. The maximum distance is that of the last segment in the spline. If there is only one segment present, the maximum distance is half the segment length. If a higher value is entered, this is ignored and the maximum value is used.
3	Gear jerk. The jerk is the change in acceleration. The value refers to the maximum value specified in the machine data. ■ 1 ... 100%

#### 10.4.3 Programming segments for a spline block

##### 10.4.3.1 Programming an SPL or SLIN segment

- Precondition**
- A program is selected.
  - Operating mode T1
  - The CP spline block fold is open.
- Procedure**
1. Move the TCP to the end point.
  2. Position the cursor in the line after which the segment is to be inserted in the spline block.
  3. Select the menu sequence **Commands > Motion > SPL or SLIN**.
  4. Set the parameters in the inline form.
  5. Press **Cmd OK**.

##### 10.4.3.2 Programming an SCIRC segment

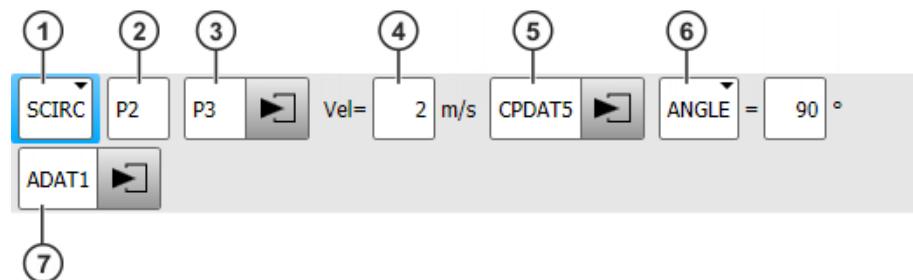
- Precondition**
- A program is selected.
  - Operating mode T1

- 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.  
([>>> 10.4.3.3 "Inline form for CP spline segment" Page 324](#))
5. Press **Teach Aux**.
6. Move the TCP to the end point.
7. Press **Cmd OK**.

#### 10.4.3.3 Inline form for CP spline segment



**Fig. 10-13: Inline form for CP spline segment**

By default, not all boxes of the inline form are displayed. The boxes can be displayed or hidden using the **Switch parameter** button.

Item	Description
1	Motion type ■ <b>SPL, SLIN or SCIRC</b>
2	Only for <b>SCIRC</b> : Point name for the auxiliary point. The system automatically generates a name. The name can be overwritten. ( <a href="#">&gt;&gt;&gt; 10.2 "Names in inline forms" Page 313</a> )
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. ( <a href="#">&gt;&gt;&gt; 10.4.3.6 "Option window "Frames" (CP and PTP spline segments)" Page 326</a> )
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. ■ <b>0.001 ... 2 m/s</b>

Item	Description
5	<p>Name for the motion data set. The system automatically generates a name. The name can be overwritten.</p> <p>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.</p> <p>Touch the arrow to edit the data. The corresponding option window is opened.</p> <p>(&gt;&gt;&gt; 10.4.3.7 "Option window "Motion parameters" (CP spline segment)" Page 327)</p>
6	<p>Circular angle</p> <p>Only available if the motion type <b>SCIRC</b> has been selected.</p> <ul style="list-style-type: none"> <li>■ - 9,999° ... + 9,999°</li> </ul> <p>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.</p>
7	<p>Name of the data set containing logic parameters. The system automatically generates a name. The name can be overwritten.</p> <p>Touch the arrow to edit the data. The corresponding option window is opened.</p> <p>(&gt;&gt;&gt; 10.4.3.9 "Option window "Logic parameters"" Page 328)</p>

#### 10.4.3.4 Programming an SPTP segment

##### Precondition

- A program is selected.
- Operating mode T1
- The PTP spline block fold is open.

##### Procedure

1. Move the TCP to the end point.
2. Position the cursor in the line after which the segment is to be inserted in the spline block.
3. Select the menu sequence **Commands > Motion > SPTP**.
4. Set the parameters in the inline form.  
(>>> 10.4.3.5 "Inline form for SPTP segment" Page 325)
5. Press **Cmd OK**.

#### 10.4.3.5 Inline form for SPTP segment

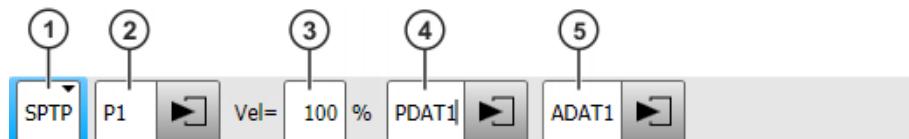


Fig. 10-14: Inline form for SPTP segment

By default, not all boxes of the inline form are displayed. The boxes can be displayed or hidden using the **Switch parameter** button.

Item	Description
1	Motion type <b>SPTP</b>
2	<p>Point name for end point. The system automatically generates a name. The name can be overwritten.</p> <p>(&gt;&gt;&gt; 10.2 "Names in inline forms" Page 313)</p> <p>Touch the arrow to edit the point data. The corresponding option window is opened.</p> <p>(&gt;&gt;&gt; 10.4.3.6 "Option window "Frames" (CP and PTP spline segments)" Page 326)</p>
3	<p>Axis velocity</p> <p>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.</p> <ul style="list-style-type: none"> <li>■ <b>1 ... 100%</b></li> </ul>
4	<p>Name for the motion data set. The system automatically generates a name. The name can be overwritten.</p> <p>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.</p> <p>Touch the arrow to edit the point data. The corresponding option window is opened.</p> <p>(&gt;&gt;&gt; 10.4.3.8 "Option window "Motion parameters" (SPTP)" Page 328)</p>
5	<p>Name of the data set containing logic parameters. The system automatically generates a name. The name can be overwritten.</p> <p>Touch the arrow to edit the data. The corresponding option window is opened.</p> <p>(&gt;&gt;&gt; 10.4.3.9 "Option window "Logic parameters"" Page 328)</p>

#### 10.4.3.6 Option window "Frames" (CP and PTP spline segments)

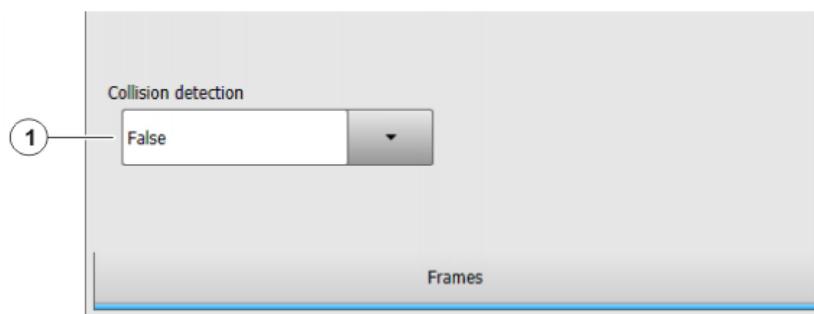


Fig. 10-15: Option window "Frames" (CP and PTP spline segments)

Item	Description
1	<ul style="list-style-type: none"> <li>■ <b>True:</b> For this motion, the robot controller calculates the axis torques. These are required for collision detection.</li> <li>■ <b>False:</b> For this motion, the robot controller does not calculate the axis torques. Collision detection is thus not possible for this motion.</li> </ul>

#### 10.4.3.7 Option window “Motion parameters” (CP spline segment)

##### Motion parameters

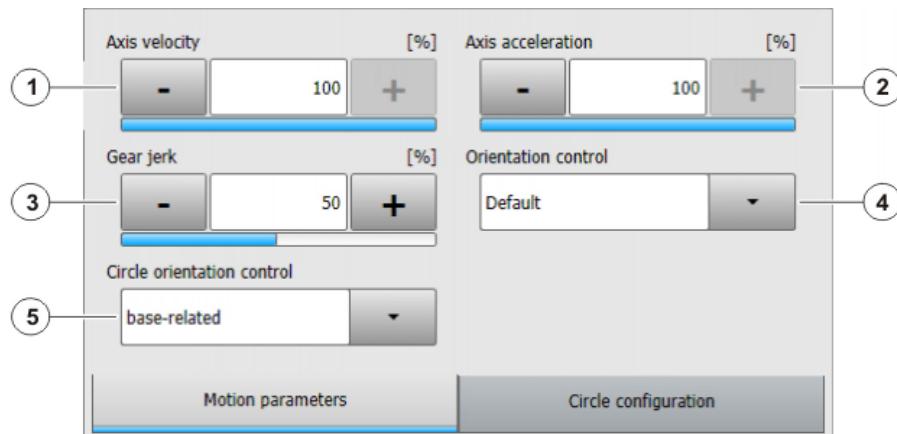


Fig. 10-16: Option window “Motion parameters” (CP spline segment)

Item	Description
1	Axis velocity. The value refers to the maximum value specified in the machine data. ■ 1 ... 100%
2	Axis acceleration. The value refers to the maximum value specified in the machine data. ■ 1 ... 100%
3	Gear jerk. The jerk is the change in acceleration. The value refers to the maximum value specified in the machine data. ■ 1 ... 100%
4	Orientation control selection.
5	Only in the case of SCIRC segments: Orientation control reference system selection.

##### Circle configuration

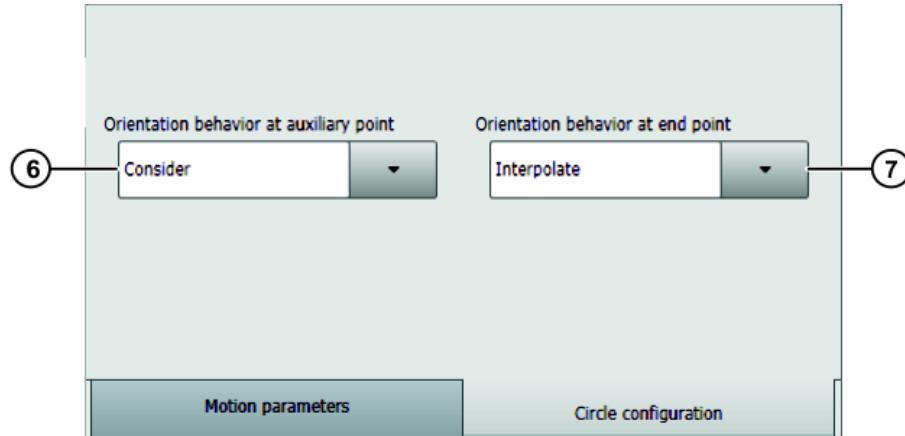


Fig. 10-17: Circle configuration (SCIRC segment)

Item	Description
6	Only in the case of SCIRC segments: Selection of orientation behavior at auxiliary point.
7	Only in the case of SCIRC segments: This box is only displayed if <b>ANGLE</b> was selected in the inline form. Selection of orientation behavior at end point.

#### 10.4.3.8 Option window “Motion parameters” (SPTP)

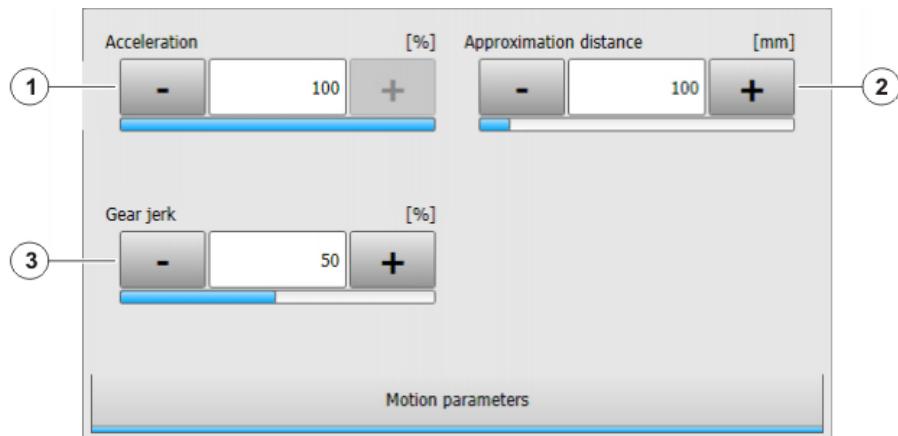


Fig. 10-18: Option window “Motion parameters” (SPTP)

Item	Description
1	Axis acceleration. The value refers to the maximum value specified in the machine data. ■ 1 ... 100%
2	This box is not available for SPTP segments. In the case of individual SPTP motions, this box is only displayed if <b>CONT</b> 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.
3	Gear jerk. The jerk is the change in acceleration.  The value refers to the maximum value specified in the machine data. ■ 1 ... 100%

#### 10.4.3.9 Option window “Logic parameters”

##### Trigger

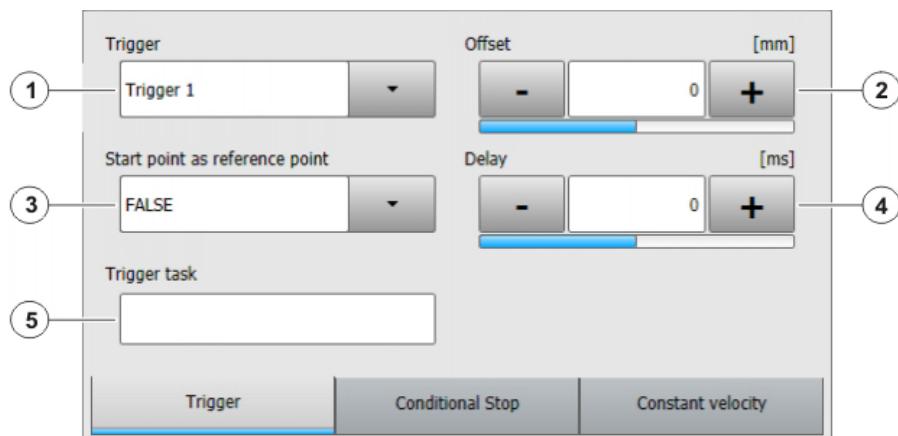


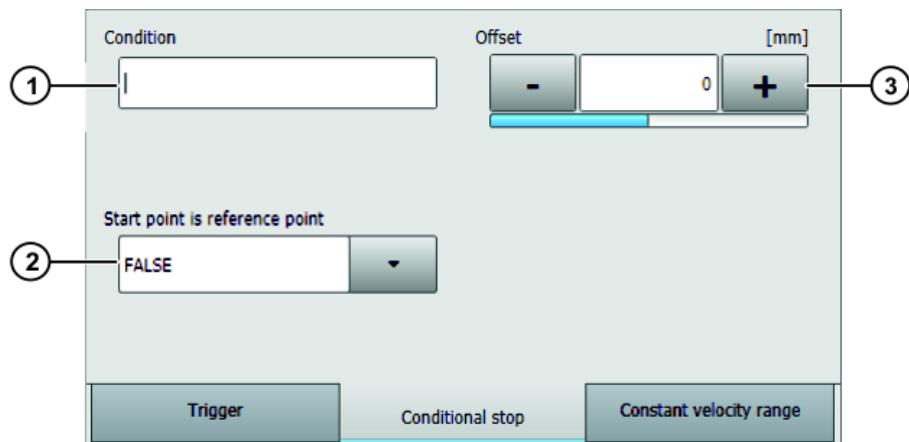
Fig. 10-19: Trigger

Item	Description
1	<p>A (further) trigger can be assigned to the motion by means of the button <b>Select action &gt; Add trigger</b>. If it is the first trigger for this motion, this command also causes the <b>Trigger</b> box to be displayed.</p> <p>A maximum of 8 triggers per motion are possible.</p> <p>(A trigger can be removed again by means of <b>Select action &gt; Remove trigger</b>.)</p>
2	<p>Reference point of the trigger</p> <ul style="list-style-type: none"> <li>■ <b>TRUE</b>: Start point</li> <li>■ <b>FALSE</b>: End point</li> </ul> <p>(&gt;&gt;&gt; 11.12.2.1 "Reference point for approximate positioning – overview" Page 442)</p>
3	<p>Spatial offset relative to the end or start point</p> <ul style="list-style-type: none"> <li>■ Negative value: Offset towards the start of the motion</li> <li>■ Positive value: Offset towards the end of the motion</li> </ul> <p>(&gt;&gt;&gt; "Max. offset" Page 440)</p> <p>The shift in space can also be taught. In this case, the box <b>Start point is reference point</b> is automatically set to <b>FALSE</b>.</p> <p>(&gt;&gt;&gt; 10.4.3.10 "Teaching the shift in space for logic parameters" Page 331)</p>
4	<p>Shift in time relative to <b>Offset</b></p> <ul style="list-style-type: none"> <li>■ Negative value: Shift towards the start of the motion.</li> <li>■ Positive value: Trigger is switched after <i>Time</i> has elapsed.</li> </ul> <p>(&gt;&gt;&gt; "Max. offset" Page 440)</p>
5	<p>Statement that is to be initiated by the trigger. The following are possible:</p> <ul style="list-style-type: none"> <li>■ Assignment of a value to a variable <b>Note:</b> There must be no runtime variable on the left-hand side of the assignment.</li> <li>■ OUT statement; PULSE statement; CYCFLAG statement</li> <li>■ Subprogram call. In this case, the priority must be specified. Example: <code>my_subprogram() PRIO = 81</code> 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: <code>PRIO = -1.</code> 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.</li> </ul>

## Conditional stop



Further information about the conditional stop can be found in this documentation.  
(>>> 10.4.5 "Conditional stop" Page 337)



**Fig. 10-20: Conditional stop**

Item	Description
1	Stop condition. The following are permitted: <ul style="list-style-type: none"> <li>■ a global Boolean variable</li> <li>■ a signal name</li> <li>■ a comparison</li> <li>■ a simple logic operation: NOT, OR, AND or EXOR</li> </ul>
2	The conditional stop can refer to either the start point or the end point of the motion. <ul style="list-style-type: none"> <li>■ <b>TRUE:</b> Start point</li> <li>■ <b>FALSE:</b> End point</li> </ul> If the reference point is approximated, the same rules apply as for the PATH trigger. ( <a href="#">&gt;&gt;&gt; 11.12.2.1 "Reference point for approximate positioning – overview" Page 442</a> )
3	The stop point can be shifted in space. For this, the desired distance from the start or end point must be specified. If no shift in space is desired, enter "0". <ul style="list-style-type: none"> <li>■ Positive value: Offset towards the end of the motion</li> <li>■ Negative value: Offset towards the start of the motion</li> </ul> There are limits to the distance the stop point can be offset. The same limits apply as for the PATH trigger. ( <a href="#">&gt;&gt;&gt; "Max. offset" Page 440</a> ) The shift in space can also be taught. In this case, the box <b>Start point is reference point</b> is automatically set to <b>FALSE</b> . ( <a href="#">&gt;&gt;&gt; 10.4.3.10 "Teaching the shift in space for logic parameters" Page 331</a> )

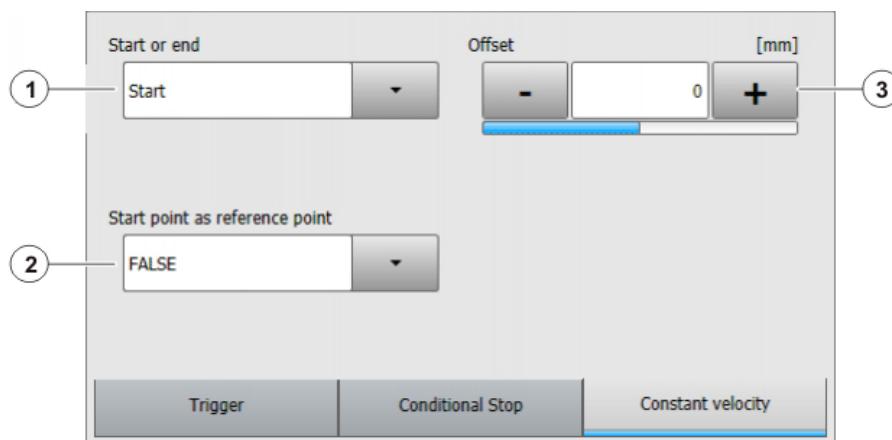
#### Constant velocity range



**Constant velocity range** is only available for CP spline segments.



Basic information about the constant velocity range can be found here:  
([>>> 10.4.6 "Constant velocity range in the CP spline block" Page 340](#))



**Fig. 10-21: Constant velocity range**

Item	Description
1	<ul style="list-style-type: none"> <li>■ <b>Start:</b> Defines the start of the constant velocity range.</li> <li>■ <b>End:</b> Defines the end of the constant velocity range.</li> </ul>
2	<p><b>Start</b> and <b>End</b> can refer to either the start point or the end point of the motion.</p> <ul style="list-style-type: none"> <li>■ <b>TRUE:</b> <b>Start</b> or <b>End</b> 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. (&gt;&gt;&gt; 11.12.2.2 "Reference point for homogenous approximate positioning" Page 443)</li> <li>■ <b>FALSE:</b> <b>Start</b> or <b>End</b> refers to the end point. If the end point is approximated, <b>Start</b> or <b>End</b> refers to the start of the approximate positioning arc.</li> </ul>
3	<p>The start or end of the constant velocity range can be shifted in space. For this, the desired distance must be specified. If no shift in space is desired, enter "0".</p> <ul style="list-style-type: none"> <li>■ Positive value: Offset towards the end of the motion</li> <li>■ Negative value: Offset towards the start of the motion</li> </ul> <p>(&gt;&gt;&gt; 10.4.6.2 "Maximum limits" Page 341)</p> <p>The shift in space can also be taught. In this case, the <b>Start point is reference point</b> box is automatically set to <b>FALSE</b>.</p> <p>(&gt;&gt;&gt; 10.4.3.10 "Teaching the shift in space for logic parameters" Page 331)</p>

#### 10.4.3.10 Teaching the shift in space for logic parameters

**Description** Shifts in space can be specified in the option window **Logic parameters** for trigger, conditional stop and constant velocity range. Instead of entering these offsets numerically, they can also be taught.

 If an offset is taught, the box **Start point is reference point** in the corresponding tab is automatically set to **FALSE**, as the taught distance refers to the end point of the motion.

**Precondition**

- A program is selected.
- Operating mode T1
- The point for which the offset is to apply has already been taught.

**Procedure**

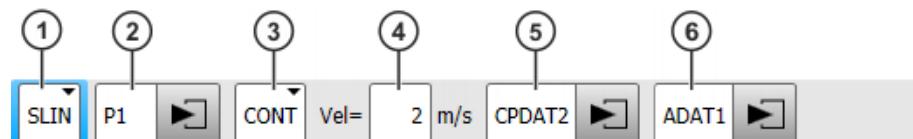
1. Move the TCP to the desired position.
2. Position the cursor in the line containing the motion instruction for which the offset is to be taught.
3. Press **Change**. The inline form for this instruction is opened.
4. Open the option window **Logic parameters** and select the required tab.
5. Press **Select action** then one of the following buttons depending on what the offset is to be taught for:
  - **Teach trigger path**
  - **Teach conditional stop path**
  - **Teach constant velocity range path**
 The distance from the end point of the current motion statement is now applied as the value for the offset.
6. Save the change by pressing **Cmd OK**.

**10.4.4 Programming individual spline motions****10.4.4.1 Programming an individual SLIN motion****Precondition**

- A program is selected.
- Operating mode T1

**Procedure**

1. Move the TCP to the end point.
2. Position the cursor in the line after which the motion is to be inserted.
3. Select **Commands > Motion > SLIN**.
4. Set the parameters in the inline form.  
(>>> 10.4.4.2 "Inline form "SLIN"" Page 332)
5. Press **Cmd OK**.

**10.4.4.2 Inline form "SLIN"****Fig. 10-22: Inline form "SLIN" (individual motion)**

<b>Item</b>	<b>Description</b>
1	Motion type <b>SLIN</b>
2	Point name for end point. The system automatically generates a name. The name can be overwritten. (>>> 10.2 "Names in inline forms" Page 313) Touch the arrow to edit the point data. The corresponding option window is opened. (>>> 10.3.7 "Option window "Frames"" Page 316)
3	<ul style="list-style-type: none"> <li>■ <b>CONT</b>: End point is approximated.</li> <li>■ <b>[blank]</b>: The motion stops exactly at the end point.</li> </ul>
4	Velocity <ul style="list-style-type: none"> <li>■ <b>0.001 ... 2 m/s</b></li> </ul>

Item	Description
5	<p>Name for the motion data set. The system automatically generates a name. The name can be overwritten.</p> <p>Touch the arrow to edit the point data. The corresponding option window is opened.</p> <p>(&gt;&gt;&gt; 10.4.4.3 "Option window "Motion parameters" (SLIN)" Page 333)</p>
6	<p>This box can be displayed or hidden by means of <b>Switch parameter</b>.</p> <p>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.</p> <p>(&gt;&gt;&gt; 10.4.3.9 "Option window "Logic parameters"" Page 328)</p>

#### 10.4.4.3 Option window "Motion parameters" (SLIN)

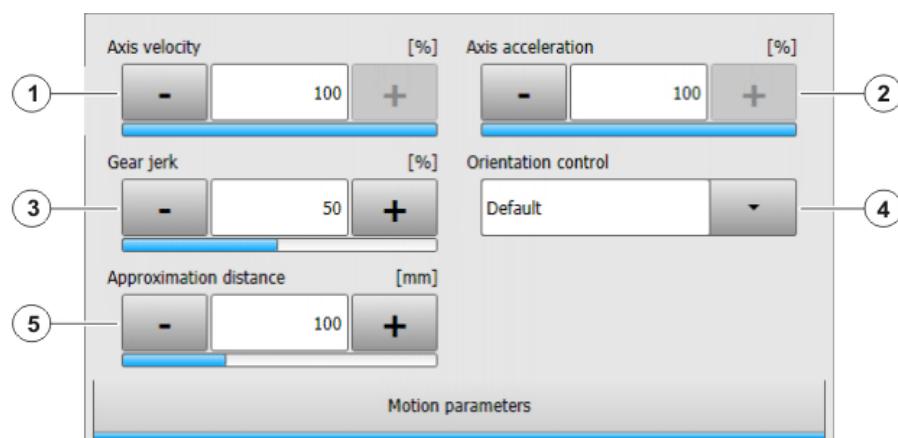


Fig. 10-23: Option window "Motion parameters" (SLIN)

Item	Description
1	Axis velocity. The value refers to the maximum value specified in the machine data. ■ 1 ... 100%
2	Axis acceleration. The value refers to the maximum value specified in the machine data. ■ 1 ... 100%
3	Gear jerk. The jerk is the change in acceleration. The value refers to the maximum value specified in the machine data. ■ 1 ... 100%
4	Orientation control selection.
5	This box is only displayed if <b>CONT</b> was selected in the inline form. Furthest distance before the end point at which approximate positioning can begin. The maximum permissible value is half the distance between the start point and the end point. If a higher value is entered, this is ignored and the maximum value is used.

#### 10.4.4.4 Programming an individual SCIRC motion

##### Precondition

- A program is selected.
- Operating mode T1

##### Procedure

1. Move the TCP to the auxiliary point.
2. Position the cursor in the line after which the motion is to be inserted.
3. Select the menu sequence **Commands > Motion > SCIRC**.
4. Set the parameters in the inline form.  
(>>> 10.4.4.5 "Inline form "SCIRC"" Page 334)
5. Press **Teach Aux**.
6. Move the TCP to the end point.
7. Press **Cmd OK**.

#### 10.4.4.5 Inline form "SCIRC"

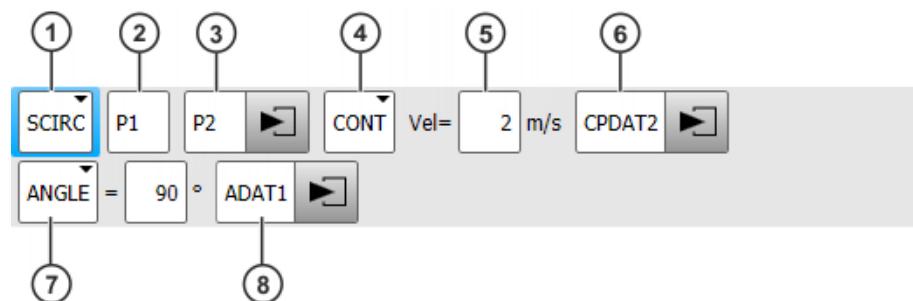


Fig. 10-24: Inline form "SCIRC" (individual motion)

Item	Description
1	Motion type <b>SCIRC</b>
2	Point name for the auxiliary point. The system automatically generates a name. The name can be overwritten. (>>> 10.2 "Names in inline forms" Page 313)
3	Point name for the end point. The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the point data. The corresponding option window is opened. (>>> 10.3.7 "Option window "Frames"" Page 316)
4	■ <b>CONT</b> : End point is approximated. ■ <b>[blank]</b> : The motion stops exactly at the end point.
5	Velocity ■ <b>0.001 ... 2 m/s</b>
6	Name for the motion data set. The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the point data. The corresponding option window is opened. (>>> 10.4.4.6 "Option window "Motion parameters" (SCIRC)" Page 335)

Item	Description
7	<p>Circular angle</p> <ul style="list-style-type: none"> <li>■ - 9,999° ... + 9,999°</li> </ul> <p>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.</p>
8	<p>This box can be displayed or hidden by means of <b>Switch parameter</b>.</p> <p>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.</p> <p>(&gt;&gt;&gt; 10.4.3.9 "Option window "Logic parameters"" Page 328)</p>

#### 10.4.4.6 Option window "Motion parameters" (SCIRC)

**Motion parameters**

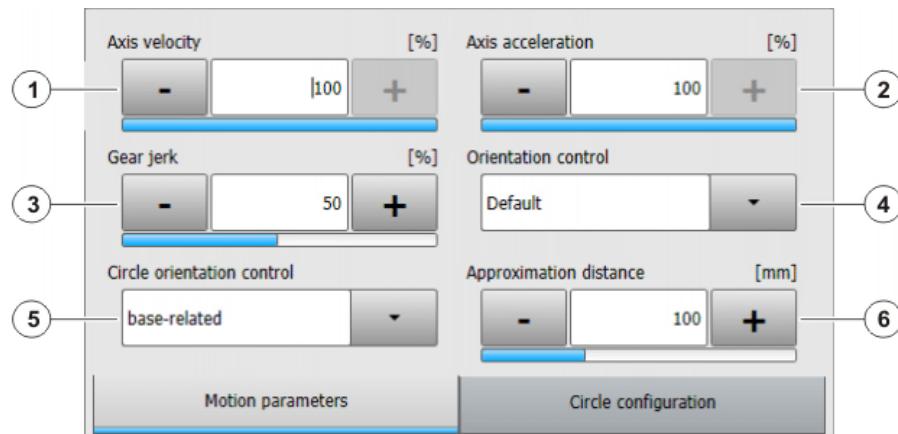


Fig. 10-25: Motion parameters (SCIRC)

Item	Description
1	<p>Axis velocity. The value refers to the maximum value specified in the machine data.</p> <ul style="list-style-type: none"> <li>■ 1 ... 100%</li> </ul>
2	<p>Axis acceleration. The value refers to the maximum value specified in the machine data.</p> <ul style="list-style-type: none"> <li>■ 1 ... 100%</li> </ul>
3	<p>Gear jerk. The jerk is the change in acceleration.</p> <p>The value refers to the maximum value specified in the machine data.</p> <ul style="list-style-type: none"> <li>■ 1 ... 100%</li> </ul>
4	Orientation control selection.
5	Orientation control reference system selection.
6	<p>This box is only displayed if <b>CONT</b> was selected in the inline form.</p> <p>Furthest distance before the end point at which approximate positioning can begin.</p> <p>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.</p>

## Circle config- uration

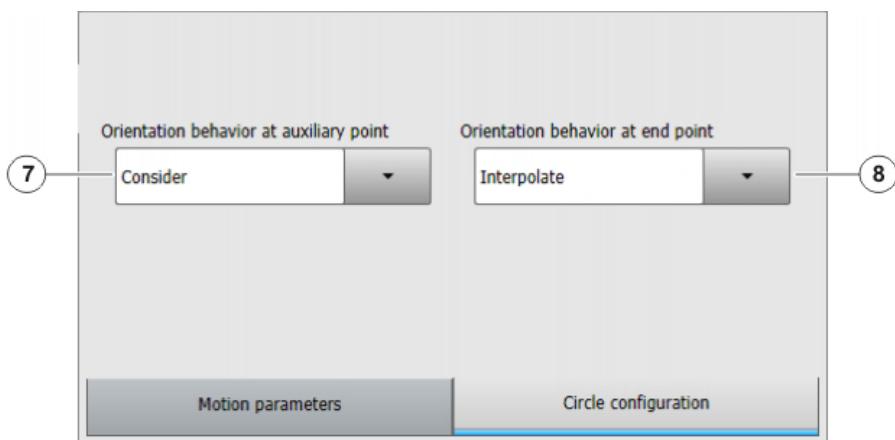


Fig. 10-26: Circle configuration (SCIRC)

Item	Description
7	Selection of orientation behavior at auxiliary point.
8	This box is only displayed if <b>ANGLE</b> was selected in the inline form. Selection of orientation behavior at end point.

### 10.4.4.7 Programming an individual SPTP motion

#### Precondition

- A program is selected.
- Operating mode T1

#### Procedure

1. Move the TCP to the end point.
2. Position the cursor in the line after which the motion is to be inserted.
3. Select **Commands > Motion > SPTP**.
4. Set the parameters in the inline form.  
(>>> 10.4.4.8 "Inline form "SPTP"" Page 336)
5. Press **Cmd OK**.

### 10.4.4.8 Inline form "SPTP"

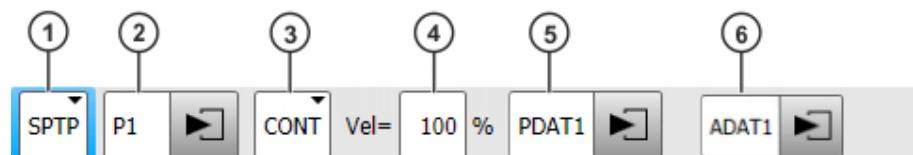


Fig. 10-27: Inline form "SPTP" (individual motion)

Item	Description
1	Motion type <b>SPTP</b>
2	Point name for end point. The system automatically generates a name. The name can be overwritten. (>>> 10.2 "Names in inline forms" Page 313) Touch the arrow to edit the point data. The corresponding option window is opened. (>>> 10.3.7 "Option window "Frames"" Page 316)
3	<ul style="list-style-type: none"> <li>■ <b>CONT</b>: End point is approximated.</li> <li>■ <b>[blank]</b>: The motion stops exactly at the end point.</li> </ul>

<b>Item</b>	<b>Description</b>
4	Velocity ■ 1 ... 100%
5	Name for the motion data set. The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the point data. The corresponding option window is opened. (>>> 10.4.3.8 "Option window "Motion parameters" (SPTP)" Page 328)
6	This box can be displayed or hidden by means of <b>Switch parameter</b> . Name of the data set containing logic parameters. The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the data. The corresponding option window is opened. (>>> 10.4.3.9 "Option window "Logic parameters"" Page 328)

#### 10.4.5 Conditional stop

- Description** The conditional stop enables the user to define a point on the path at which the robot stops if a certain condition is met. This point is called the "stop point". As soon as the condition is no longer met, the robot resumes its motion.
- During the runtime, the robot controller calculates the latest point at which the robot must brake in order to be able to stop at the stop point. From this point (braking point) onwards, it monitors whether or not the condition is met.
- If the condition is met at the braking point, the robot brakes in order to stop at the stop point.  
If, however, the condition then switches back to "not met" before the stop point is reached, the robot accelerates again and does not stop.
  - If the condition is not met at the braking point, the robot motion is continued without braking.
- Essentially, any number of conditional stops can be programmed. A maximum of 10 "braking point → stop point" paths may overlap, however.
- While the robot is braking, the robot controller displays the following message in T1/T2 mode: *Conditional stop active (line {Line number})*.
- (>>> 10.4.5.2 "Stop condition: example and braking characteristics" Page 339)
- Programming** Programming with KRL syntax:
- using the statement STOP WHEN PATH
- Programming with inline forms:
- In the spline block (CP and PTP) or in the individual spline block:  
in the option window "**Logic parameters**"
  - Before a spline block (CP and PTP):  
via the inline form **Spline Stop Condition**

##### 10.4.5.1 Inline form "Spline Stop Condition"

This inline form may only be used before a spline block. There may be other statements between the inline form and the spline block, but no motion instructions.



**Fig. 10-28: Inline form “Spline Stop Condition”**

Item	Description
1	<p>Point to which the conditional stop refers</p> <ul style="list-style-type: none"> <li>■ With ONSTART: last point before the spline block</li> <li>■ Without ONSTART: last point in the spline block</li> </ul> <p>If the spline is approximated, the same rules apply as for the PATH trigger.</p> <p><b>Note:</b> Information about approximate positioning with the PATH trigger is contained in the Operating and Programming Instructions for System Integrators.</p> <p>ONSTART can be set or removed using the <b>Toggle OnStart</b> button.</p>
2	<p>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".</p> <ul style="list-style-type: none"> <li>■ Positive value: Offset towards the end of the motion</li> <li>■ Negative value: Offset towards the start of the motion</li> </ul> <p>There are limits to the distance the stop point can be offset. The same limits apply as for the PATH trigger.</p> <p>The shift in space can also be taught.</p> <p>(&gt;&gt;&gt; "Teach path" Page 338)</p>
3	<p>Stop condition</p> <p>The following are permitted:</p> <ul style="list-style-type: none"> <li>■ a global Boolean variable</li> <li>■ a signal name</li> <li>■ a comparison</li> <li>■ a simple logic operation: NOT, OR, AND or EXOR</li> </ul>

#### Teach path

Button	Description
<b>Teach path</b>	<p>If an offset is desired, it is not necessary to enter the value into the inline form numerically; the offset can also be taught. This is done via <b>Teach path</b>.</p> <p>If an offset is taught, ONSTART is automatically removed, if set in the inline form, as the taught distance always refers to the end point of the motion.</p> <p>The teaching sequence is the same as that for the option window <b>Logic parameters</b>.</p> <p>(&gt;&gt;&gt; 10.4.3.10 "Teaching the shift in space for logic parameters" Page 331)</p>

#### 10.4.5.2 Stop condition: example and braking characteristics

##### Example

The indentations are not present by default and have been inserted here for greater clarity.

```

PTP P0 Vel=100 % PDAT1 Tool[1] Base[0]

SPLINE S1 Vel=2 m/s CPDAT1 Tool[1] Base[0]

SPL P1 ADAT1

STOP WHEN PATH = 50 IF $in[77]==FALSE

SPL XP1

SPL P2

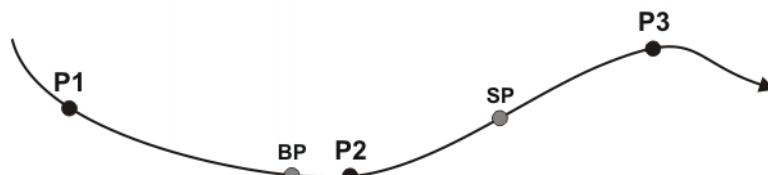
SPL P3

ENDSPLINE

```

**Fig. 10-29: Inline programming example (folds expanded)**

Line	Description
4	If the input \$IN[77] is FALSE, the robot stops 50 mm after P2 and waits until \$IN[77] is TRUE.



**Fig. 10-30: Example of STOP WHEN PATH**

Point	Description
<b>BP</b>	<b>Braking Point:</b> 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 <b>BP</b> depends on the velocity and the override setting and cannot be identified by the operator.
<b>SP</b>	<b>Stop Point</b> The distance <b>P2 → SP</b> is 50 mm long.

##### Braking characteristics

Situation at BP	Behavior of the robot
\$IN[77] == FALSE	The robot brakes and stops at <b>SP</b> .
\$IN[77] == TRUE	The robot does not brake and does not stop at <b>SP</b> . The program is executed as if the STOP WHEN PATH statement were not present.

Situation at BP	Behavior of the robot
1. \$IN[77] == FALSE at BP. 2. The input switches to TRUE between BP and SP.	1. The robot brakes at BP. 2. If the input is TRUE, the robot accelerates again and does not stop at SP.
1. \$IN[77] == TRUE at BP. 2. The input switches to FALSE between BP and SP.	1. The robot does not brake at BP. 2. If the input is FALSE, the robot stops with a path-maintaining EMERGENCY STOP and comes to a standstill at an unpredictable point.

If the stop condition is not met until the robot has already passed **BP**, it is too late to stop at **SP** with a normal braking ramp. In this case, the robot stops with a path-maintaining EMERGENCY STOP and comes to a standstill at an unpredictable point.

- If the EMERGENCY STOP causes the robot to stop after **SP**, the program cannot be resumed until the stop condition is no longer met.
- If the path-maintaining EMERGENCY STOP causes the robot to stop before **SP**, the following occurs when the program is resumed:
  - If the stop condition is no longer met, the robot resumes its motion.
  - If the stop condition is still met, the robot moves as far as **SP** and stops there.

#### 10.4.6 Constant velocity range in the CP spline block

##### Description

In a CP spline block, a range can be defined in which the robot maintains the programmed velocity constant where possible. This range is called the “constant velocity range”.

- 1 constant velocity range can be defined per CP spline block.
- A constant velocity range is defined by a start statement and an end statement.
- The range cannot extend beyond the spline block.
- There is no lower limit to the size of the range.

If it is not possible to maintain the programmed velocity constant, the robot controller indicates this by means of a message during program execution.

##### Constant velocity range over several segments:

A constant velocity range can extend over several segments with different programmed velocities. In this case, the lowest of the velocities is valid for the whole range.

Even in the segments with a higher programmed velocity, the motion is executed with the lowest velocity in this case. No message is generated indicating that the velocity has not been maintained. This only occurs if the lowest velocity cannot be maintained.

##### Programming

A constant velocity range can be programmed in the following ways:

- If programming with KRL syntax: by means of the statement CONST\_VEL
- If programming with inline forms:

The start or end of the range is stored in the corresponding CP segment in the option window **Logic parameters**.

### 10.4.6.1 Block selection to the constant velocity range

#### Description

If a block selection to a constant velocity range is carried out, the robot controller ignores it and generates a corresponding message. The motions are executed as if no constant velocity range were programmed.

A block selection to the path section defined by the offset values is considered as a block selection to the constant velocity range. The motion blocks in which the start and end of the range are programmed, however, are irrelevant.

### 10.4.6.2 Maximum limits

**If the start or end point of the spline block is an exact positioning point:**

- The constant velocity range starts at the start point at the earliest.
- The constant velocity range ends at the end point at the latest.

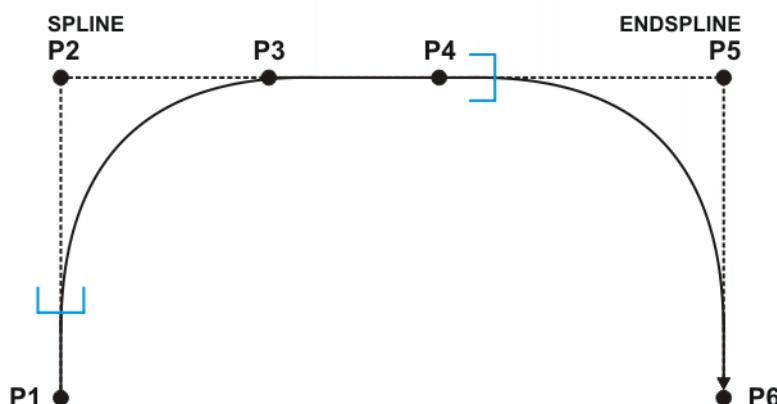
If the offset value is such that these limits would be exceeded, the robot controller automatically reduces the offset and generates the following message: *CONST\_VEL {Start/End} = {Offset} cannot be implemented; {New offset} will be used.*

The robot controller reduces the offset far enough to create a range in which the constant programmed velocity can be maintained. In other words, it does not necessarily shift the limit exactly to the start or end point of the spline block, but possibly further inwards.

The same message is generated if the range is already in the spline block beforehand, but the defined velocity cannot be maintained due to the offset. In this case, once again, the robot controller reduces the offset.

**If the start or end point of the spline block is approximated:**

- The constant velocity range starts at the start of the approximate positioning arc of the start point at the earliest.
- The constant velocity range ends at the start of the approximate positioning arc of the end point at the latest.



**Fig. 10-31: Maximum limits for approximated SPLINE/ENDSPLINE**

If the offset is such that these limits would be exceeded, the robot controller automatically sets the limit to the start of the corresponding approximate positioning arc. It does not generate a message.

## 10.5 Displaying the distance between points

#### Precondition

- Program is selected or open.
- T1 or T2 operating mode
- "Expert" user group

- |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Procedure</b> | <ol style="list-style-type: none"><li>1. Select the points (= the motion blocks) for which the distance is to be displayed. Multiple consecutive blocks can also be selected.</li><li>2. Select the menu sequence <b>Edit &gt; Marked region &gt; Cart. distance</b>.<br/>A window opens. It displays the following information:<ul style="list-style-type: none"><li>■ The Cartesian coordinates of the first selected point</li><li>■ The Cartesian coordinates of the last selected point</li><li>■ The distance between the coordinates in millimeters and degrees</li><li>■ The distance between the TCP position at the first and last point in millimeters and degrees</li></ul></li><li>3. Select other points if required.</li><li>4. Press <b>Refresh</b> to update the display.</li></ol> |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

## 10.6 Modifying programmed motions

### 10.6.1 Modifying motion parameters

- |                     |                                                                                                                                                                                                                                                                                                       |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Precondition</b> | <ul style="list-style-type: none"><li>■ A program is selected.</li><li>■ Operating mode T1</li></ul>                                                                                                                                                                                                  |
| <b>Procedure</b>    | <ol style="list-style-type: none"><li>1. Position the cursor in the line containing the instruction that is to be changed.</li><li>2. Press <b>Change</b>. The inline form for this instruction is opened.</li><li>3. Modify parameters.</li><li>4. Save changes by pressing <b>Cmd Ok</b>.</li></ol> |

### 10.6.2 Modifying blocks of motion parameters

- |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Precondition</b> | <ul style="list-style-type: none"><li>■ “Expert” user group</li><li>■ A program is selected.</li><li>■ Operating mode T1</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Procedure</b>    | <ol style="list-style-type: none"><li>1. Select the motion instructions to be modified. (Only consecutive motion instructions can be modified as a block.)</li><li>2. Press <b>Change</b>. The inline form of the first selected motion block opens.</li><li>3. Modify parameters.</li><li>4. Press <b>Cmd OK</b>. The changes will be applied to the selected motion blocks where possible.<br/>Some changes will not be applied in every motion block, e.g. it is not possible to apply the PTP parameter <b>Velocity</b> in a LIN motion block.</li></ol> |

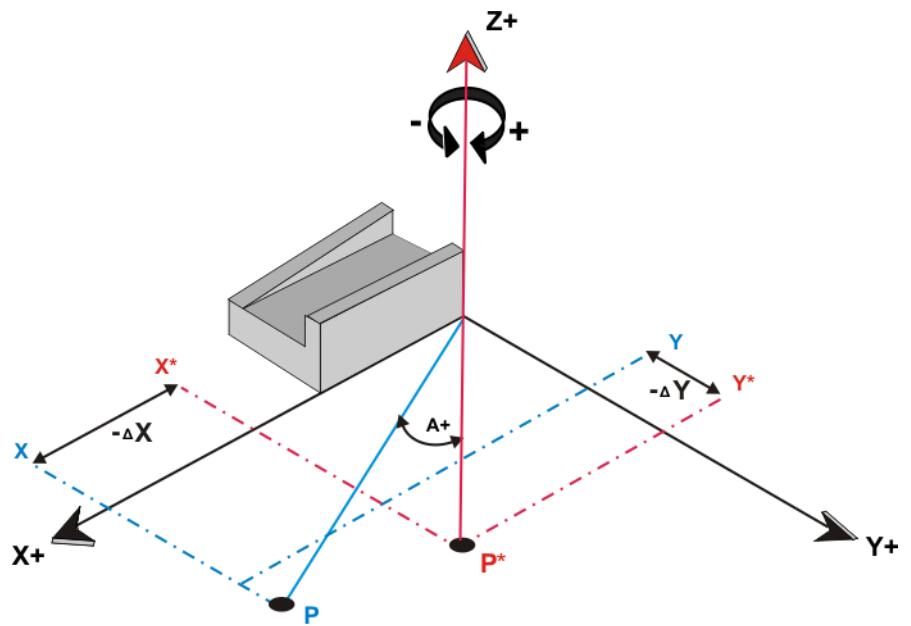
### 10.6.3 Re-teaching a point

- |                     |                                                                                                                                                                                                                                                                              |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b>  | 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.                                                                                                                           |
| <b>Precondition</b> | <ul style="list-style-type: none"><li>■ A program is selected.</li><li>■ Operating mode T1</li></ul>                                                                                                                                                                         |
| <b>Procedure</b>    | <ol style="list-style-type: none"><li>1. Move the TCP to the desired position.</li><li>2. Position the cursor in the line containing the motion instruction that is to be changed.</li><li>3. Press <b>Change</b>. The inline form for this instruction is opened.</li></ol> |

4. For PTP and LIN motions: Press **Touch Up** to accept the current position of the TCP as the new end point.
- For CIRC motions:
- Press **Teach Aux** to accept the current position of the TCP as the new auxiliary point.
  - Or press **Teach End** to accept the current position of the TCP as the new end point.
5. Confirm the request for confirmation with **Yes**.
  6. Save change by pressing **Cmd Ok**.

#### **10.6.4 Transforming blocks of coordinates**

<b>Precondition</b>	<ul style="list-style-type: none"> <li>■ "Expert" user group</li> <li>■ A program is selected.</li> <li>■ Operating mode T1</li> </ul>
<b>Procedure</b>	<ol style="list-style-type: none"> <li>1. Select the motion instructions to be modified. (Only consecutive motion instructions can be modified as a block.)</li> <li>2. Select the menu sequence <b>Edit &gt; Marked region</b>. Select transformation type. The corresponding window is opened. (&gt;&gt;&gt; 10.6.4.1 "'Axis mirroring' window" Page 346) (&gt;&gt;&gt; 10.6.4.2 "'Transform - Axis Specific" window" Page 347) (&gt;&gt;&gt; 10.6.4.3 "'Transform - Cartesian Base" window" Page 348)</li> <li>3. Enter values for the transformation and press <b>Calculate</b>.</li> </ol>
<b>Overview</b>	The following transformation types are available: <ul style="list-style-type: none"> <li>■ <b>Transform - Cartesian base</b></li> <li>■ <b>Transform - Cartesian tool</b></li> <li>■ <b>Transform - Cartesian World</b></li> <li>■ <b>Transform - Axis-specific</b></li> <li>■ <b>Axis mirroring</b></li> </ul>
<b>Transform - Base</b>	<b>Transform - Cartesian Base:</b> The transformation refers to the current BASE coordinate system.



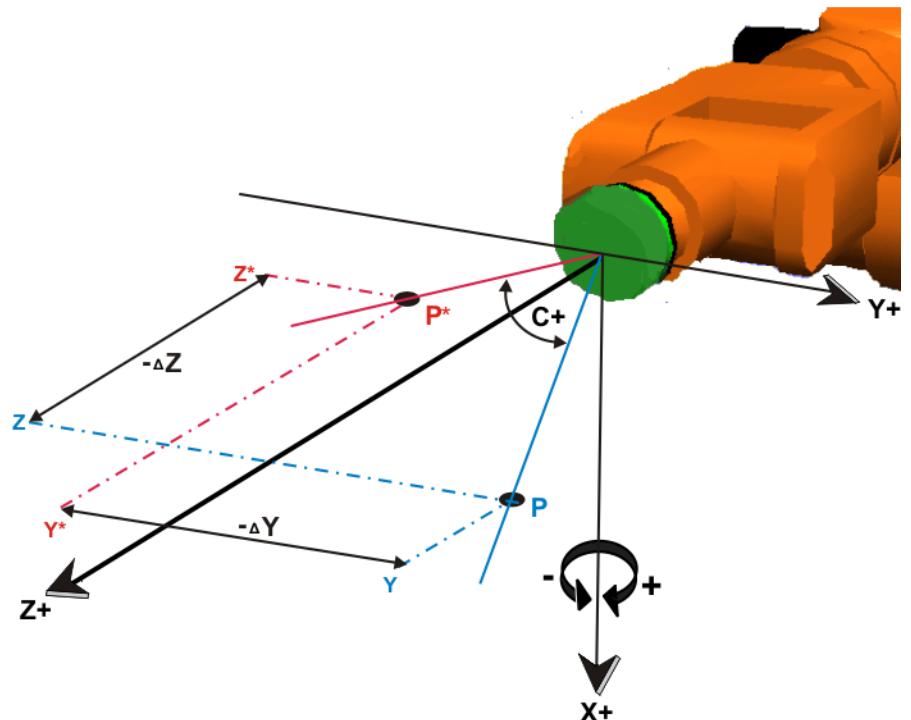
**Fig. 10-32: Transform - Cartesian Base**

Point P is offset by  $\Delta X$  and  $\Delta Y$  in the negative direction. The new position of the point is  $P^*$ .

**Transform - TCP**

**Transform - Cartesian TCP:**

The transformation refers to the current TOOL coordinate system.



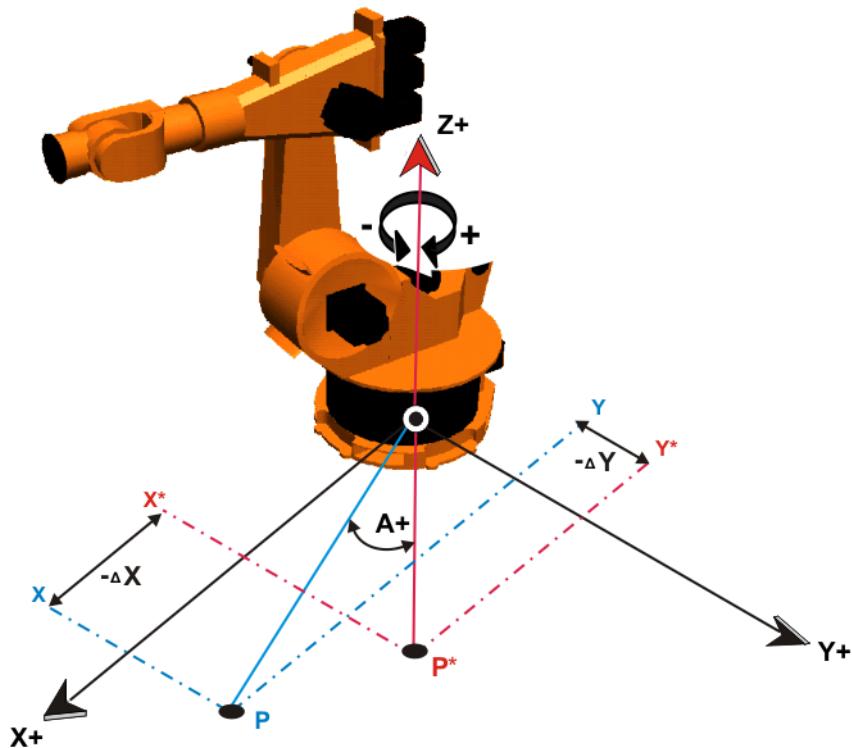
**Fig. 10-33: Transform - Cartesian TCP**

Point P is offset by  $\Delta Z$  and  $\Delta Y$  in the negative direction. The new position of the point is  $P^*$ .

**Transform - World**

**Transform - Cartesian World:**

The transformation is relative to the WORLD coordinate system.



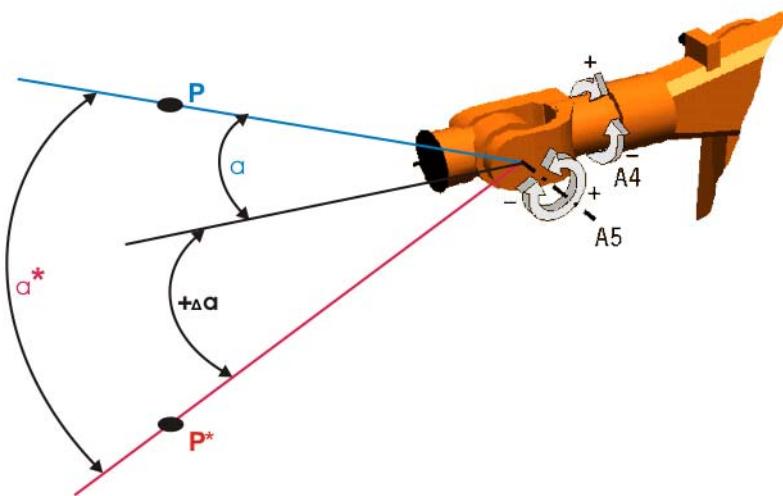
**Fig. 10-34: Transform - Cartesian World**

Point P is offset by  $\Delta X$  and  $\Delta Y$  in the negative direction. The new position of the point is  $P^*$ .

**Transform - Axis Specific**

**Transform - Axis Specific:**

The transformation is axis-specific.



**Fig. 10-35: Transform - Axis Specific**

Axis A5 is rotated by the angle  $\Delta a$ . The new position of point P is  $P^*$ .

**Mirroring**

**Mirroring:**

Mirroring in the XY plane of the ROBROOT coordinate system.

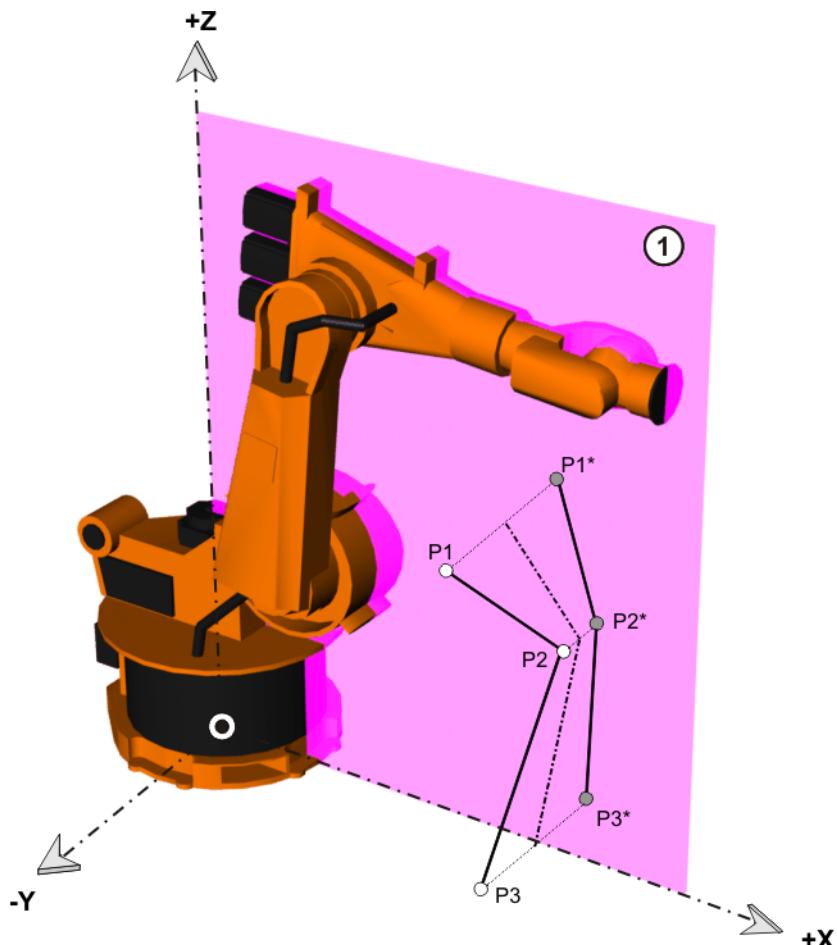


Fig. 10-36: Mirroring

Points P1, P2 and P3 are mirrored in the XY plane (1). The new positions of the points are P1\*, P2\* and P3\*.

#### 10.6.4.1 “Axis mirroring” window

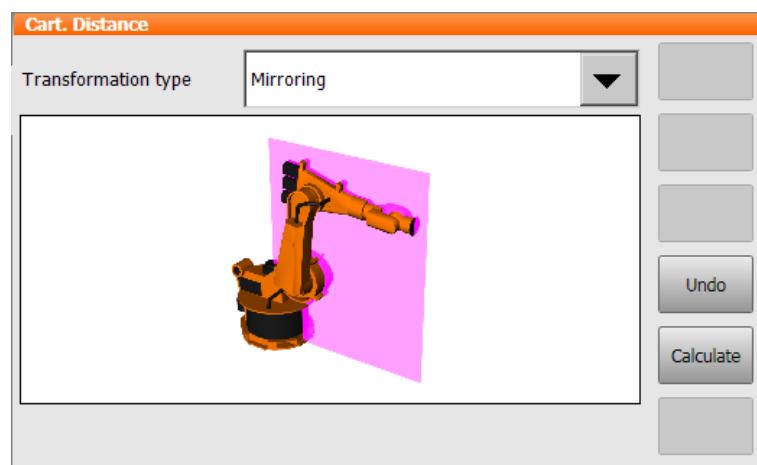
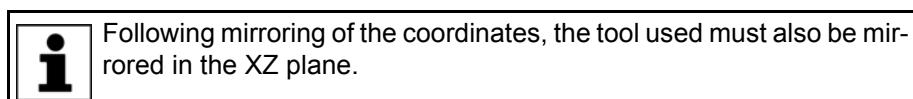


Fig. 10-37: Mirroring

No values need to be entered in this window. Pressing the **Calculate** button mirrors the point coordinates in the XZ plane of the ROBROOT coordinate system.



The following buttons are available:

Button	Description
<b>Calculate</b>	Mirrors the coordinates of the selected motion points in the XZ plane, converts the coordinates to axis angles and applies the new values.
<b>Undo</b>	Undoes the mirroring and restores the old point data.

Only selected points with a complete E6POS definition are copied. This includes, for example, all those that were generated via inline forms during programming. Points without a complete E6POS definition are ignored during the point offset.

#### 10.6.4.2 “Transform - Axis Specific” window

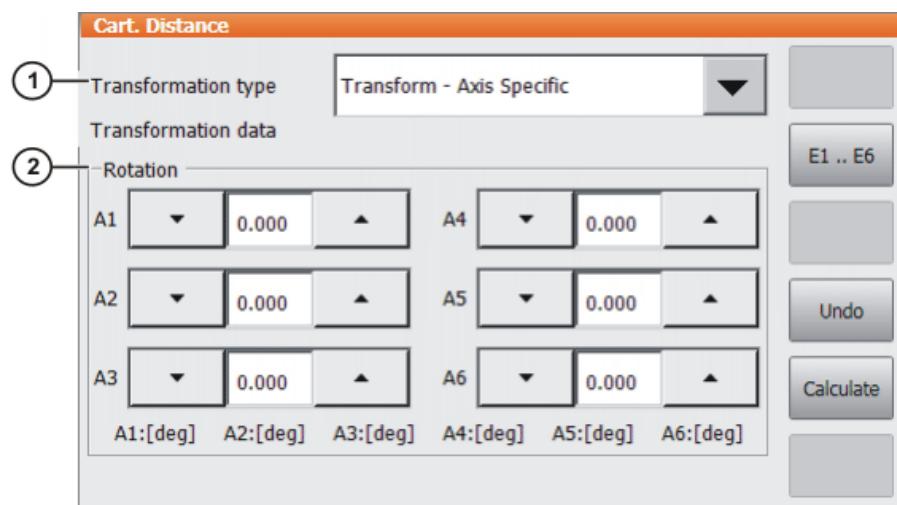


Fig. 10-38: Point transformation - axis-specific

Item	Description
1	Selection of the transformation type
2	<b>Rotation</b> group: input boxes for the position offset of axes A1 ... A6 <ul style="list-style-type: none"> <li>■ Range of values: Dependent on the configuration of the axis-specific workspaces</li> </ul> <b>E1 .. E6</b> switches to the <b>External axes</b> group: input boxes for the position offset of axes E1 ... E6 <p><b>Note:</b> Values can only be entered for configured axes.</p>

The following buttons are available:

Button	Description
<b>E1 .. E6/A1 .. A6</b>	Toggles between the <b>Rotation</b> and <b>External axes</b> groups.

Button	Description
<b>Undo</b>	Undoes the transformation and restores the old point data.
<b>Calculate</b>	Calculates the point transformation and applies it to all selected motion points.  If the transformation would cause a point to be situated outside the configured workspace, the point is not transformed.

Only selected points with a complete E6POS definition are copied. This includes, for example, all those that were generated via inline forms during programming. Points without a complete E6POS definition are ignored during the point offset.

#### 10.6.4.3 “Transform - Cartesian Base” window

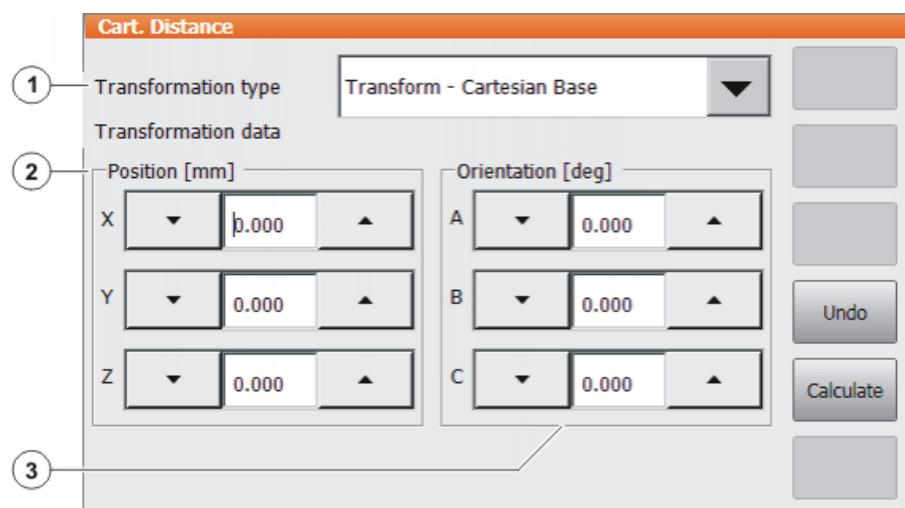


Fig. 10-39: Point transformation - Cartesian

Item	Description
1	Selection of the transformation type
2	<b>Position</b> group: input boxes for the point transformation in the X, Y, Z direction <ul style="list-style-type: none"> <li>Range of values: Dependent on the configuration of the Cartesian workspaces</li> </ul>
3	<b>Orientation</b> group: input boxes for the transformation of the A, B, C orientation <ul style="list-style-type: none"> <li>Range of values: Dependent on the configuration of the Cartesian workspaces</li> </ul>

The following buttons are available:

Button	Description
<b>Undo</b>	Undoes the transformation and restores the old point data.
<b>Calculate</b>	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 Programming logic instructions

### 10.7.1 Inputs/outputs

#### Digital inputs/outputs

The robot controller can manage up to 8192 digital inputs and 8192 digital outputs. 4096 inputs/outputs are available by default.

#### Analog inputs/outputs

The robot controller can manage 32 analog inputs and 32 analog outputs.

The inputs/outputs are managed via the following system variables:

	Inputs	Outputs
Digital	\$IN[1] ... \$IN[8192]	\$OUT[1] ... \$OUT[8192]
Analog	\$ANIN[1] ... \$ANIN[32]	\$ANOUT[1] ... \$ANOUT[32]

\$ANIN[...] indicates the input voltage, adapted to the range between -1.0 and +1.0. The actual voltage depends on the settings of the analog module.

\$ANOUT[...] can be used to set an analog voltage. \$ANOUT[...] can have values from -1.0 to +1.0 written to it. The voltage actually generated depends on the settings of the analog module. If an attempt is made to set voltages outside the range of values, the robot controller displays the following message: *Limit {Signal name}*

### 10.7.2 Setting a digital output - OUT

#### Precondition

- A program is selected.
- Operating mode T1

#### Procedure

1. Position the cursor in the line after which the logic instruction is to be inserted.
2. Select the menu sequence **Commands > Logic > OUT > OUT**.
3. Set the parameters in the inline form.  
(>>> 10.7.3 "Inline form "OUT"" Page 349)
4. Save instruction with **Cmd Ok**.

### 10.7.3 Inline form "OUT"

The instruction sets a digital output.

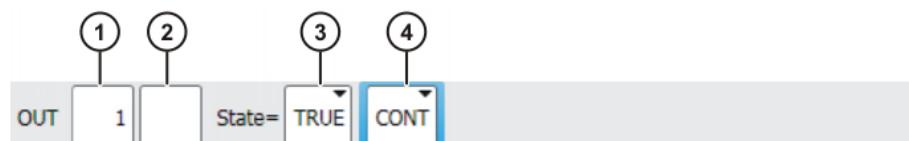


Fig. 10-40: Inline form "OUT"

Item	Description
1	Number of the output
2	If a name exists for the output, this name is displayed. User group "Expert" or higher: A name can be entered by pressing <b>Long text</b> . The name is freely selectable.
3	State to which the output is switched <ul style="list-style-type: none"> <li>■ <b>TRUE</b></li> <li>■ <b>FALSE</b></li> </ul>
4	<ul style="list-style-type: none"> <li>■ <b>CONT</b>: Execution in the advance run</li> <li>■ <b>[Empty box]</b>: Execution with advance run stop</li> </ul>

#### 10.7.4 Setting a pulse output - PULSE

- Precondition**
- A program is selected.
  - Operating mode T1
- Procedure**
1. Position the cursor in the line after which the logic instruction is to be inserted.
  2. Select the menu sequence **Commands > Logic > OUT > PULSE**.
  3. Set the parameters in the inline form.  
(>>> 10.7.5 "Inline form "PULSE"" Page 350)
  4. Save instruction with **Cmd Ok**.

#### 10.7.5 Inline form "PULSE"

The instruction sets a pulse of a defined length.

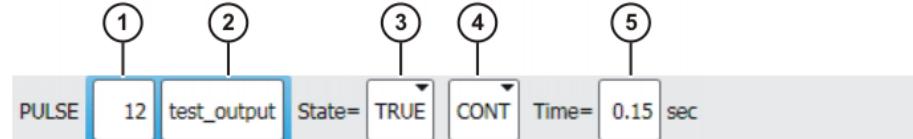


Fig. 10-41: Inline form "PULSE"

Item	Description
1	Number of the output
2	If a name exists for the output, this name is displayed. User group "Expert" or higher: A name can be entered by pressing <b>Long text</b> . The name is freely selectable.
3	State to which the output is switched <ul style="list-style-type: none"> <li>■ <b>TRUE</b>: "High" level</li> <li>■ <b>FALSE</b>: "Low" level</li> </ul>
4	<ul style="list-style-type: none"> <li>■ <b>CONT</b>: Execution in the advance run</li> <li>■ <b>[Empty box]</b>: Execution with advance run stop</li> </ul>
5	Length of the pulse <ul style="list-style-type: none"> <li>■ <b>0.10 ... 3.00 s</b></li> </ul>

## 10.7.6 Setting an analog output - ANOUT

### Precondition

- A program is selected.
- Operating mode T1

### Procedure

1. Position the cursor in the line after which the instruction is to be inserted.
2. Select **Commands > Analog output > Static or Dynamic**.
3. Set the parameters in the inline form.
 

(>>> 10.7.7 "Inline form "ANOUT" (static)" Page 351)  
 (>>> 10.7.8 "Inline form "ANOUT" (dynamic)" Page 351)
4. Save instruction with **Cmd Ok**.

## 10.7.7 Inline form "ANOUT" (static)

This instruction sets a static analog output. The voltage is set to a fixed level by means of a factor. The actual voltage level depends on the analog module used. For example, a 10 V module with a factor of 0.5 provides a voltage of 5 V.

ANOUT triggers an advance run stop.



**Fig. 10-42: 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)

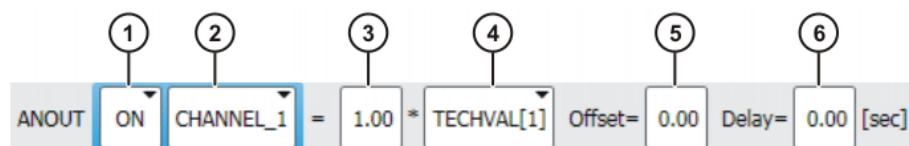
## 10.7.8 Inline form "ANOUT" (dynamic)

This instruction activates or deactivates a dynamic analog output.

A maximum of 4 dynamic analog outputs can be activated at any one time.  
ANOUT triggers an advance run stop.

The voltage is determined by a factor. The actual voltage level depends on the following values:

- Velocity or function generator  
For example, a velocity of 1 m/s with a factor of 0.5 results in a voltage of 5 V.
- Offset  
For example, an offset of +0.15 for a voltage of 0.5 V results in a voltage of 6.5 V.



**Fig. 10-43: Inline form "ANOUT" (dynamic)**

Item	Description
1	Activation or deactivation of the analog output <ul style="list-style-type: none"> <li>■ ON</li> <li>■ OFF</li> </ul>
2	Number of the analog output <ul style="list-style-type: none"> <li>■ CHANNEL_1 ... CHANNEL_32</li> </ul>
3	Factor for the voltage <ul style="list-style-type: none"> <li>■ 0 ... 10 (intervals: 0.01)</li> </ul>
4	<ul style="list-style-type: none"> <li>■ VEL_ACT: The voltage is dependent on the velocity.</li> <li>■ TECHVAL[1] ... TECHVAL[6]: The voltage is controlled by a function generator.</li> </ul>
5	Value by which the voltage is increased or decreased <ul style="list-style-type: none"> <li>■ -1 ... +1 (intervals: 0.01)</li> </ul>
6	Time by which the output signal is delayed (+) or brought forward (-) <ul style="list-style-type: none"> <li>■ -0.2 ... +0.5 s</li> </ul>

### 10.7.9 Programming a wait time - WAIT

- Precondition**
- A program is selected.
  - Operating mode T1
- Procedure**
1. Position the cursor in the line after which the logic instruction is to be inserted.
  2. Select the menu sequence **Commands > Logic > WAIT**.
  3. Set the parameters in the inline form.  
*(>>> 10.7.10 "Inline form "WAIT"" Page 352)*
  4. Save instruction with **Cmd Ok**.

### 10.7.10 Inline form "WAIT"

WAIT can be used to program a wait time. The robot motion is stopped for a programmed time. WAIT always triggers an advance run stop.



Fig. 10-44: Inline form "WAIT"

Item	Description
1	Wait time <ul style="list-style-type: none"> <li>■ <math>\geq 0 \text{ s}</math></li> </ul>

### 10.7.11 Programming a signal-dependent wait function - WAITFOR

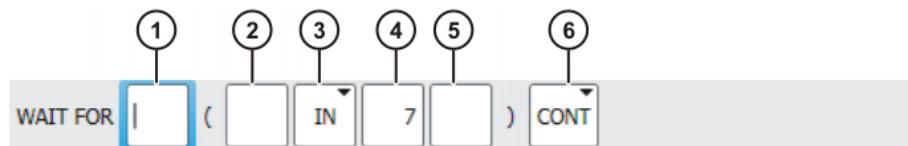
- Precondition**
- A program is selected.
  - Operating mode T1

- Procedure**
1. Position the cursor in the line after which the logic instruction is to be inserted.
  2. Select the menu sequence **Commands > Logic > WAITFOR**.
  3. Set the parameters in the inline form.  
(**>>> 10.7.12 "Inline form "WAITFOR""** Page 353)
  4. Save instruction with **Cmd Ok**.

### 10.7.12 Inline form "WAITFOR"

The instruction sets a signal-dependent wait function.

If required, several signals (maximum 12) can be linked. If a logic operation is added, boxes are displayed in the inline form for the additional signals and links.



**Fig. 10-45: Inline form "WAITFOR"**

Item	Description
1	Add external logic operation. The operator is situated between the bracketed expressions. <ul style="list-style-type: none"> <li>■ <b>AND</b></li> <li>■ <b>OR</b></li> <li>■ <b>EXOR</b></li> </ul> Add NOT. <ul style="list-style-type: none"> <li>■ <b>NOT</b></li> <li>■ <b>[blank]</b></li> </ul> Enter the desired operator by means of the corresponding button.
2	Add internal logic operation. The operator is situated inside a bracketed expression. <ul style="list-style-type: none"> <li>■ <b>AND</b></li> <li>■ <b>OR</b></li> <li>■ <b>EXOR</b></li> </ul> Add NOT. <ul style="list-style-type: none"> <li>■ <b>NOT</b></li> <li>■ <b>[blank]</b></li> </ul> Enter the desired operator by means of the corresponding button.
3	Signal for which the system is waiting <ul style="list-style-type: none"> <li>■ <b>IN</b></li> <li>■ <b>OUT</b></li> <li>■ <b>CYCFLAG</b></li> <li>■ <b>TIMER</b></li> <li>■ <b>FLAG</b></li> </ul>
4	Number of the signal

Item	Description
5	If a name exists for the signal, this name is displayed. User group "Expert" or higher: A name can be entered by pressing <b>Long text</b> . The name is freely selectable.
6	<ul style="list-style-type: none"> <li>■ <b>CONT</b>: Execution in the advance run</li> <li>■ <b>[Empty box]</b>: Execution with advance run stop</li> </ul>

### 10.7.13 Switching on the path - SYN OUT

#### Precondition

- A program is selected.
- Operating mode T1

#### Procedure

1. Position the cursor in the line after which the logic instruction is to be inserted.
2. Select the menu sequence **Commands > Logic > OUT > SYN OUT**.
3. Set the parameters in the inline form.
 

(>>> 10.7.14 "Inline form "SYN OUT", option "START/END"" Page 354)  
   (>>> 10.7.15 "Inline form "SYN OUT", option "PATH"" Page 357)
4. Save instruction with **Cmd Ok**.

### 10.7.14 Inline form "SYN OUT", option "START/END"

The switching action can be triggered relative to the start or end point of the motion. The switching action can be delayed or brought forward. The motion can be a LIN, CIRC or PTP motion.

Possible applications include:

- Closing or opening the weld gun during spot welding
- Switching the welding current on/off during arc welding
- Starting or stopping the flow of adhesive in bonding or sealing applications.

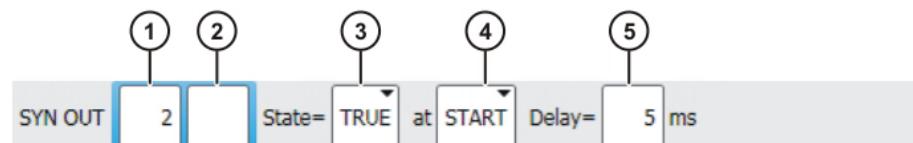


Fig. 10-46: Inline form "SYN OUT", option "START/END"

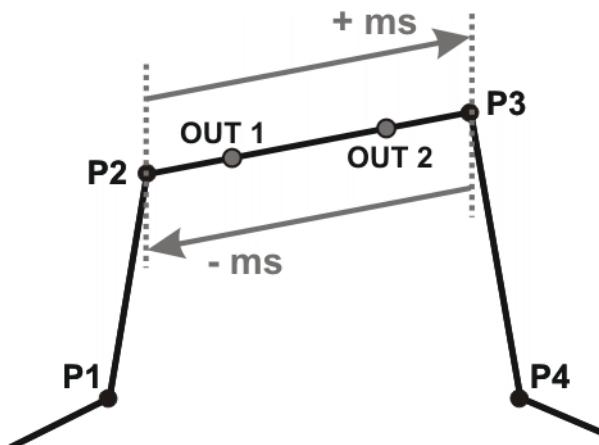
Item	Description
1	Number of the output
2	If a name exists for the output, this name is displayed. User group "Expert" or higher: A name can be entered by pressing <b>Long text</b> . The name is freely selectable.
3	State to which the output is switched <ul style="list-style-type: none"> <li>■ <b>TRUE</b></li> <li>■ <b>FALSE</b></li> </ul>

Item	Description
4	Point to which SYN OUT refers: <ul style="list-style-type: none"> <li>■ <b>START:</b> Start point of the motion</li> <li>■ <b>END:</b> End point of the motion</li> </ul>
5	Switching action delay <ul style="list-style-type: none"> <li>■ <b>-1,000 ... +1,000 ms</b></li> </ul> <p><b>Note:</b> The time specification is absolute. In other words, the switching point varies according to the velocity of the robot.</p>

**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. 10-47**

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
```

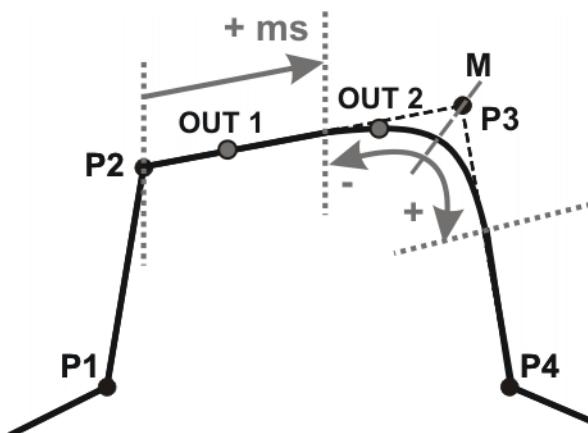


Fig. 10-48

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
```

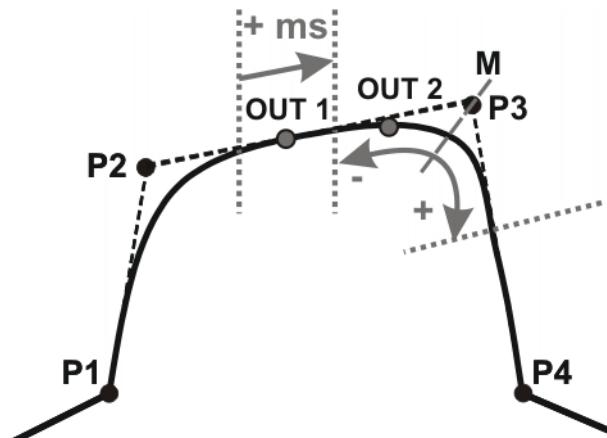


Fig. 10-49

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.

### 10.7.15 Inline form "SYN OUT", option "PATH"

The switching action refers to the end point of the motion. The switching action can be shifted in space and delayed or brought forward. The motion can be a LIN or CIRC motion. It must not be a PTP motion.

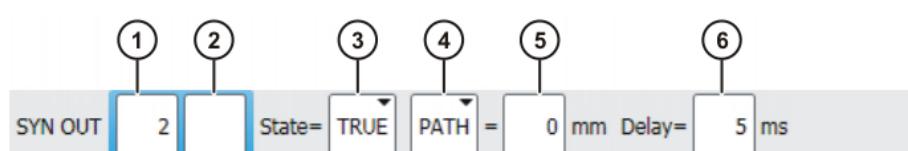


Fig. 10-50: Inline form "SYN OUT", option "PATH"

Item	Description
1	Number of the output
2	If a name exists for the output, this name is displayed. User group "Expert" or higher: A name can be entered by pressing <b>Long text</b> . The name is freely selectable.
3	State to which the output is switched <ul style="list-style-type: none"> <li>■ <b>TRUE</b></li> <li>■ <b>FALSE</b></li> </ul>
4	■ <b>PATH</b> : SYN OUT refers to the end point of the motion.
5	This box is only displayed if <b>PATH</b> has been selected. Distance from the switching point to the end point <ul style="list-style-type: none"> <li>■ <b>-2,000 ... +2,000 mm</b></li> </ul>
6	Switching action delay <ul style="list-style-type: none"> <li>■ <b>-1,000 ... +1,000 ms</b></li> </ul> <b>Note:</b> 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
```

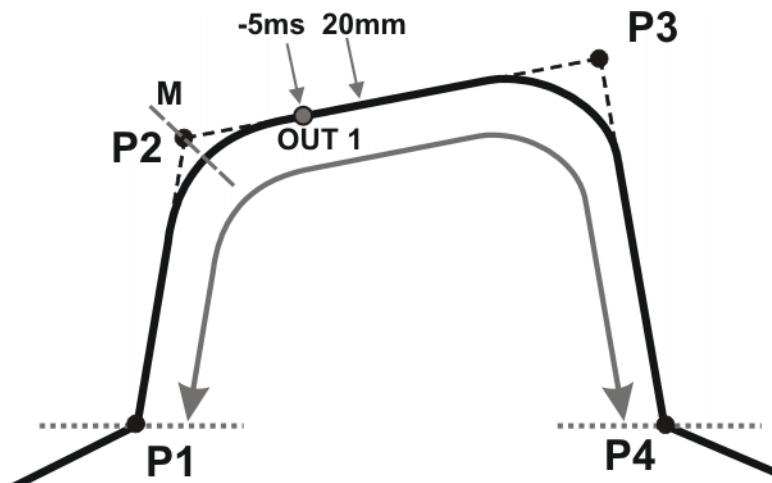


Fig. 10-51

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
```

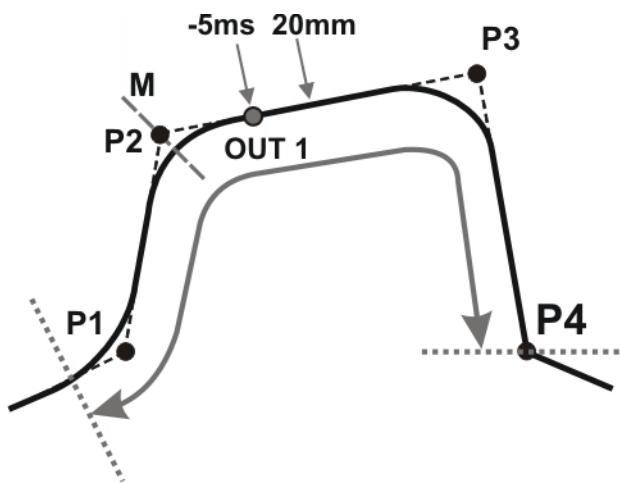


Fig. 10-52

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.

## 10.7.16 Setting a pulse on the path - SYN PULSE

### Precondition

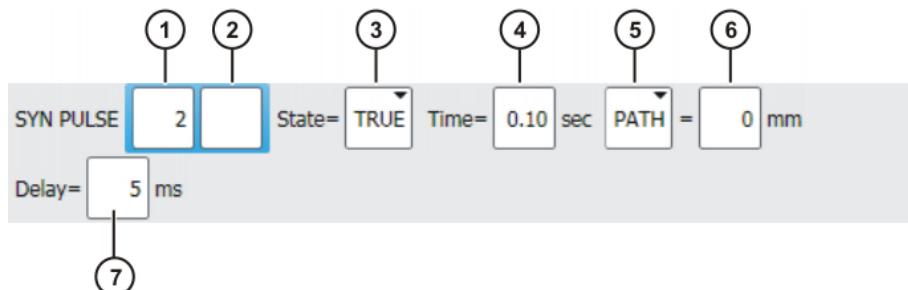
- A program is selected.
- Operating mode T1

### Procedure

1. Position the cursor in the line after which the logic instruction is to be inserted.
2. Select the menu sequence **Commands > Logic > OUT > SYN PULSE**.
3. Set the parameters in the inline form.  
(>>> 10.7.17 "Inline form "SYN PULSE"" Page 359)
4. Save instruction with **Cmd Ok**.

## 10.7.17 Inline form "SYN PULSE"

SYN PULSE can be used to trigger a pulse at the start or end point of the motion. The pulse can be shifted in time and/or space, i.e. it does not have to be triggered exactly at the point, but can also be triggered before or after it.



**Fig. 10-53: Inline form "SYN PULSE"**

Item	Description
1	Number of the output
2	If a name exists for the output, this name is displayed. User group "Expert" or higher: A name can be entered by pressing <b>Long text</b> . The name is freely selectable.
3	State to which the output is switched <ul style="list-style-type: none"> <li>■ <b>TRUE</b></li> <li>■ <b>FALSE</b></li> </ul>
4	Duration of the pulse <ul style="list-style-type: none"> <li>■ <b>0.1 ... 3 s</b></li> </ul>

Item	Description
5	<p>Point to which SYN PULSE refers:</p> <ul style="list-style-type: none"> <li>■ <b>START</b>: Start point of the motion</li> <li>■ <b>END</b>: End point of the motion</li> </ul> <p>See SYN OUT for examples and switching limits. (&gt;&gt;&gt; 10.7.14 "Inline form "SYN OUT", option "START/END"" Page 354)</p> <ul style="list-style-type: none"> <li>■ <b>PATH</b>: SYN PULSE refers to the end point. An offset in space is also possible.</li> </ul> <p>See SYN OUT for examples and switching limits. (&gt;&gt;&gt; 10.7.15 "Inline form "SYN OUT", option "PATH"" Page 357)</p>
6	<p>Distance from the switching point to the end point</p> <ul style="list-style-type: none"> <li>■ <b>-2,000 ... +2,000 mm</b></li> </ul> <p>This box is only displayed if <b>PATH</b> has been selected.</p>
7	<p>Switching action delay</p> <ul style="list-style-type: none"> <li>■ <b>-1,000 ... +1,000 ms</b></li> </ul> <p><b>Note:</b> The time specification is absolute. The switching point varies according to the velocity of the robot.</p>

### 10.7.18 Modifying a logic instruction

- Precondition**
- A program is selected.
  - Operating mode T1
- Procedure**
1. Position the cursor in the line containing the instruction that is to be changed.
  2. Press **Change**. The inline form for this instruction is opened.
  3. Change the parameters.
  4. Save changes by pressing **Cmd Ok**.

## 11 Programming for user group (KRL syntax)

### 11.1 Instructions for programming

**NOTICE**

When programming motions, it must be ensured that the energy supply system is not wound up or damaged during program execution.

**NOTICE**

In the case of programs with the following axis motions or positions, the film of lubricant on the gear units of the axes may break down:

- Motions <3°
- Oscillating motions
- Areas of gear units permanently facing upwards

It must be ensured that the gear units have a sufficient supply of oil. For this, in the case of oscillating motions or short motions (<3°), programming must be carried out in such a way that the affected axes regularly move more than 40° (e.g. once per cycle).

In the case of areas of gear units permanently facing upwards, sufficient oil supply must be achieved by programming re-orientations of the in-line wrist. In this way, the oil can reach all areas of the gear units by means of gravity. Required frequency of re-orientations:

- With low loads (gear unit temperature <+35 °C): daily
- With medium loads (gear unit temperature +35 °C to 55 °C): hourly
- With heavy loads (gear unit temperature >+55 °C): every 10 minutes

Failure to observe this precaution may result in damage to the gear units.

### 11.2 Overview of KRL syntax

Variables and declarations	
DECL	(>>> 11.5.1 "DECL: declaring variables, arrays and constants" Page 369)
ENUM	(>>> 11.5.2 "ENUM: defining an enumeration type" Page 370)
STRUC	(>>> 11.5.3 "STRUC: defining a structure type" Page 371)

Motion programming	
PTP	(>>> 11.6.1 "PTP: motion programming" Page 373)
LIN	(>>> 11.6.2 "LIN: motion programming" Page 373)
CIRC	(>>> 11.6.3 "CIRC: motion programming" Page 374)
PTP_REL	(>>> 11.6.4 "PTP_REL: relative motion programming" Page 375)
LIN_REL	(>>> 11.6.5 "LIN_REL: relative motion programming" Page 376)
CIRC_REL	(>>> 11.6.6 "CIRC_REL: relative motion programming" Page 377)
SPLINE ... ENDSPLINE	(>>> 11.7.1 "SPLINE ... ENDSPLINE: programming a CP spline block" Page 380)
PTP_SPLINE ... ENDSPLINE	(>>> 11.7.2 "PTP_SPLINE ... ENDSPLINE: programming a PTP spline block" Page 382)

<b>Motion programming</b>	
SLIN	(>>> 11.7.3 "SLIN: motion programming" Page 383)
SCIRC	(>>> 11.7.4 "SCIRC: motion programming" Page 383)
SPL	(>>> 11.7.5 "SPL: motion programming" Page 384)
SPTP	(>>> 11.7.6 "SPTP: motion programming" Page 385)
SLIN_REL	(>>> 11.7.7 "SLIN_REL: relative motion programming" Page 386)
SCIRC_REL	(>>> 11.7.8 "SCIRC_REL: relative motion programming" Page 387)
SPL_REL	(>>> 11.7.9 "SPL_REL: relative motion programming" Page 388)
SPTP_REL	(>>> 11.7.10 "SPTP_REL: relative motion programming" Page 389)
TIME_BLOCK	(>>> 11.7.12 "TIME_BLOCK: programming a time block for spline" Page 392)
CONST_VEL	(>>> 11.7.13 "CONST_VEL: programming a constant velocity range for spline motions" Page 395)
STOP WHEN PATH	(>>> 11.7.14 "STOP WHEN PATH: programming a conditional stop for spline" Page 397)

<b>Program execution control</b>	
CONTINUE	(>>> 11.8.1 "CONTINUE: preventing an advance run stop" Page 399)
EXIT	(>>> 11.8.2 "EXIT: exiting a loop" Page 400)
FOR ... TO ... ENDFOR	(>>> 11.8.3 "FOR ... TO ... ENDFOR: programming a counting loop" Page 400)
GOTO	(>>> 11.8.4 "GOTO: jump to position in the program" Page 401)
HALT	(>>> 11.8.5 "HALT: stopping a program" Page 402)
IF ... THEN ... ENDIF	(>>> 11.8.6 "IF ... THEN ... ENDIF: programming a conditional branch" Page 402)
LOOP ... ENDLOOP	(>>> 11.8.7 "LOOP ... ENDLOOP: programming an endless loop" Page 403)
ON_ERROR_PROCEED	(>>> 11.8.8 "ON_ERROR_PROCEED: suppressing a runtime error message" Page 403)
REPEAT ... UNTIL	(>>> 11.8.9 "REPEAT ... UNTIL: programming a post-test loop" Page 408)
SWITCH ... CASE ... ENDSWITCH	(>>> 11.8.10 "SWITCH ... CASE ... ENDSWITCH: programming a multiple branch" Page 409)
WAIT FOR ...	(>>> 11.8.11 "WAIT FOR ... : waiting until a condition has been met" Page 410)
WAIT SEC ...	(>>> 11.8.12 "WAIT SEC ... : programming a wait time" Page 411)
WHILE ... ENDWHILE	(>>> 11.8.13 "WHILE ... ENDWHILE: programming a rejecting loop" Page 411)

<b>Inputs/outputs</b>	
ANIN	(>>> 11.9.1 "ANIN: cyclically reading an analog input" Page 412)
ANOUT	(>>> 11.9.2 "ANOUT: writing cyclically to an analog output" Page 413)

<b>Inputs/outputs</b>	
PULSE	(>>> 11.9.3 "PULSE: setting a pulse output" Page 414)
SIGNAL	(>>> 11.9.4 "SIGNAL: signal declaration for inputs/outputs" Page 418)

<b>Subprograms and functions</b>	
DEFFCT ... ENDFCT	(>>> 11.10.3 "DEFFCT ... ENDFCT: defining a function" Page 420)
RETURN	(>>> 11.10.4 "RETURN: jump back to the calling program" Page 420)

<b>Interrupt programming</b>	
INTERRUPT ... DECL ... WHEN ... DO	(>>> 11.11.1 "INTERRUPT ... DECL ... WHEN ... DO ... : declaring an interrupt" Page 425)
INTERRUPT ON/OFF	(>>> 11.11.2 "INTERRUPT ON/OFF: activating or deactivating an interrupt" Page 428)
INTERRUPT DISABLE/ENABLE	(>>> 11.11.3 "INTERRUPT DISABLE/ENABLE: disabling or enabling an interrupt" Page 429)
BRAKE	(>>> 11.11.4 "BRAKE: stopping the robot from an interrupt program" Page 432)
RESUME	(>>> 11.11.5 "RESUME: aborting interrupt programs" Page 433)

<b>Path-related switching actions (=Trigger)</b>	
TRIGGER WHEN DISTANCE	(>>> 11.12.1 "TRIGGER WHEN DISTANCE" Page 435)
TRIGGER WHEN PATH	(>>> 11.12.2 "TRIGGER WHEN PATH" Page 438)

<b>Communication</b>	
(>>> 11.13 "Communication" Page 446)	

<b>Operators</b>	
Arithmetic operators	(>>> 11.14.1 "Arithmetic operators" Page 447)
Geometric operator	(>>> 11.14.2 "Geometric operator" Page 448)
Relational operators	(>>> 11.14.3 "Relational operators" Page 451)
Logic operators	(>>> 11.14.4 "Logic operators" Page 452)
Bit operators	(>>> 11.14.5 "Bit operators" Page 452)
Priority of the operators	(>>> 11.14.6 "Priority of the operators" Page 454)

<b>System functions</b>	
DELETE_BACKWARD_BUFFER()	(>>> 11.16.1 "DELETE_BACKWARD_BUFFER()" Page 457)
FORWARD()	(>>> 11.16.2 "FORWARD()" Page 458)
INV_POS()	(>>> 11.16.3 "INV_POS()" Page 459)
INVERSE()	(>>> 11.16.4 "INVERSE()" Page 459)
ROB_STOP()	(>>> 11.16.5 "ROB_STOP() and ROB_STOP_RELEASE()" Page 461)
SET_BRAKE_DELAY()	(>>> 11.16.6 "SET_BRAKE_DELAY()" Page 462)
VARSTATE()	(>>> 11.16.7 "VARSTATE()" Page 465)

**Mathematical standard functions**

(=&gt;&gt; 11.15 "Mathematical standard functions" Page 455)

**Manipulating string variables**

(=&gt;&gt; 11.17 "Editing string variables" Page 467)

**11.3 Symbols and fonts**

The following symbols and fonts are used in the syntax descriptions:

Syntax element	Representation
KRL code	<ul style="list-style-type: none"> <li>■ Courier font</li> <li>■ Upper-case letters</li> </ul> <p>Examples: GLOBAL; ANIN ON; OFFSET</p>
Elements that must be replaced by program-specific entries	<ul style="list-style-type: none"> <li>■ Italics</li> <li>■ Upper/lower-case letters</li> </ul> <p>Examples: <i>Distance</i>; <i>Time</i>; <i>Format</i></p>
Optional elements	<ul style="list-style-type: none"> <li>■ In angle brackets</li> </ul> <p>Example: &lt;STEP Increment&gt;</p>
Elements that are mutually exclusive	<ul style="list-style-type: none"> <li>■ Separated by the " " symbol</li> </ul> <p>Example: IN  OUT</p>

**11.4 Important KRL terms****11.4.1 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 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"  
By default, the SRC file and the DAT file are displayed individually. They can be edited individually.

**11.4.2 Naming conventions and keywords****Names**

Examples of names in KRL: variable names, program names, point names

- Names in KRL can have a maximum length of 24 characters.  
In some cases, less than 24 characters are allowed, e.g. a maximum of 23 characters in inline forms.

- Names in KRL can consist of letters (A-Z), numbers (0-9) and the signs “\_” and “\$”.
- Names in KRL must not begin with a number.
- Names in KRL must not be keywords.

Other restrictions may apply in the case of inline forms in technology packages.



The names of all system variables begin with the “\$” sign. To avoid confusion, do not begin the names of user-defined variables with this sign.

## Keywords

Keywords are sequences of letters having a fixed meaning. They must not be used in programs in any way other than with this meaning. No distinction is made between uppercase and lowercase letters. A keyword remains valid irrespective of the way in which it is written.

**Example:** The sequence of letters CASE is an integral part of the KRL syntax SWITCH ... CASE ... ENDSWITCH. For this reason, CASE must not be used in any other way, e.g. as a variable name.

The system distinguishes between reserved and non-reserved keywords:

- Reserved keywords  
These may only be used with their defined meaning.
- Non-reserved keywords  
With non-reserved keywords, the meaning is restricted to a particular context. Outside of this context, a non-reserved keyword is interpreted by the compiler as a name.



In practice, it is not helpful to distinguish between reserved and non-reserved keywords. To avoid error messages or compiler problems, keywords are thus never used other than with their defined meaning.

### Overview of important keywords:

All elements of the KRL syntax described in this documentation that are not program-specific are keywords.

The following important keywords are worth a particular mention:

AXIS	ENDFCT
BOOL	ENDFOR
CHAR	ENDIF
CAST_FROM	ENDLOOP
CAST_TO	ENDSWITCH
CCLOSE	ENDWHILE
CHANNEL	EXT
CIOCTL	EXTFCT
CONFIRM	FALSE
CONST	FRAME
COPEN	GLOBAL
CREAD	INT
CWRITE	MAXIMUM
DEF	MINIMUM
DEFAULT	POS
DEFDAT	PRIO
DEFFCT	PUBLIC

E6AXIS	SREAD
E6POS	SWRITE
END	REAL
ENDDAT	TRUE

### 11.4.3 Data types

#### Overview

There are 2 kinds of data types:

- User-defined data types

User-defined data types are always derived from the data types ENUM or STRUC.

- Predefined data types, e.g.:

- Simple data types
- Data types for motion programming

The following simple data types are predefined:

Data type	Keyword	Description
Integer	INT	Integer ■ -2 <sup>31</sup> -1 ... 2 <sup>31</sup> -1 Examples: 1; 32; 345
Real	REAL	Floating-point number ■ +1.1E-38 ... +3.4E+38 Examples: 1.43; 38.50; 300.25
Boolean	BOOL	Logic state ■ TRUE ■ FALSE
Character	CHAR	1 character ■ ASCII character Examples: "A"; "1"; "q"

The following data types for motion programming are predefined:

#### Structure type AXIS

A1 to A6 are angle values (rotational axes) or translation values (translational axes) for the axis-specific movement of robot axes 1 to 6.

```
STRUC AXIS REAL A1, A2, A3, A4, A5, A6
```

#### Structure type E6AXIS

E1 to E6 are angle values or translation values of the external axes 7 to 12.

```
STRUC E6AXIS REAL A1, A2, A3, A4, A5, A6, E1, E2, E3, E4, E5, E6
```

#### Structure type FRAME

X, Y and Z are space coordinates, while A, B and C are the orientation of the coordinate system.

```
STRUC FRAME REAL X, Y, Z, A, B, C
```

#### Structure types POS and E6POS

S (Status) and T (Turn) define axis positions unambiguously.

```
STRUC POS REAL X, Y, Z, A, B, C, INT S, T
```

```
STRUC E6POS REAL X, Y, Z, A, B, C, E1, E2, E3, E4, E5, E6, INT S, T
```

#### 11.4.4 Areas of validity

##### Local

Data object	Area of validity
Variable Signal	<ul style="list-style-type: none"> <li>■ 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”.</li> <li>■ If the object was defined in a DAT file: It is valid in the SRC file that belongs to the DAT file.</li> </ul>
Constant	Valid in the module to which the data list in which the constant was declared belongs.
User-defined data type	<ul style="list-style-type: none"> <li>■ If the data type was defined in an SRC file: it is valid at, or below, the program level in which it was declared.</li> <li>■ If the data type was defined in a DAT file: it is valid in the SRC file that belongs to the DAT file.</li> </ul>
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.

- ([">>>> 11.4.4.1 "Making subprograms, functions and interrupts available globally" Page 367](#))
- ([">>>> 11.4.4.2 "Making variables, constants, signals and user data types available globally" Page 368](#))

If there are local and global objects with the same name, the compiler uses the local object within its area of validity.

##### 11.4.4.1 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)
```

### 11.4.4.2 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 DEF DAT 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
...
```

## 11.4.5 Constants

The value of a constant can no longer be modified during program execution after initialization. Constants can be used to prevent a value from being changed accidentally during program execution.

Constants must be declared and, at the same time, initialized in a data list. The data type must be preceded by the keyword CONST.

```
DECL <GLOBAL> CONST Data type Variable name = Value
```

## 11.5 Variables and declarations

### 11.5.1 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 <i>Data type</i> is a predefined data type. If <i>Data type</i> is a user-defined data type, then DECL is obligatory.
GLOBAL	(>>> 11.4.4 "Areas of validity" Page 367)
CONST	The keyword CONST must only be used in data lists.
<i>Data type</i>	Specification of the desired data type
<i>Name</i>	Name of the object (variable, array or constant) that is being declared.
<i>Dimension</i>	Type: INT  <i>Dimension</i> defines the number of array elements for the dimension in question. Arrays have a minimum of 1 and a maximum of 3 dimensions.
<i>Value</i>	The data type of <i>Value</i> must be compatible with <i>Data type</i> , 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"

```

### 11.5.2 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	(>>> 11.4.4 "Areas of validity" Page 367)  <b>Note:</b> Data types defined using the keyword GLOBAL must not be used in \$CONFIG.DAT.

Element	Description
<i>NameEnum-Type</i>	Name of the new enumeration type. Recommendation: For user-defined data types, assign names ending in _TYPE, to distinguish them from variable names.
<i>Constant</i>	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
...
```

**11.5.3 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	(>>> 11.4.4 "Areas of validity" Page 367) <b>Note:</b> 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}}

```

**11.6 Motion programming: PTP, LIN, CIRC****11.6.1 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
<i>End point</i>	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.
<i>Approximation</i> ( <i>blending</i> )	<i>Approximation</i> causes the end point to be approximated. At the same time, this parameter defines the earliest point at which the approximate positioning can begin.  (>>> 11.6.7 "Approximation parameters for PTP, LIN CIRC and ..._REL" Page 378)

**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}
```

**11.6.2 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
	<i>End point</i>	<p>Type: POS, E6POS, FRAME</p> <p>If not all components of the end point are specified, the controller takes the values of the previous position for the missing components.</p> <p>The Status and Turn specifications for an end point of type POS or E6POS are disregarded.</p> <p>The coordinates refer to the BASE coordinate system.</p>
	<i>Approximation (blending)</i>	<p><i>Approximation</i> causes the end point to be approximated. At the same time, this parameter defines the earliest point at which the approximate positioning can begin.</p> <p>(&gt;&gt;&gt; 11.6.7 "Approximation parameters for PTP, LIN CIRC and ..._REL" Page 378)</p>

**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}
```

### 11.6.3 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
	<i>Auxiliary point</i>	<p>Type: POS, E6POS, FRAME</p> <p>If not all components of the auxiliary point are specified, the controller takes the values of the previous position for the missing components.</p> <p>The orientation angles and the Status and Turn specifications for an auxiliary point are always disregarded.</p> <p>The auxiliary point cannot be approximated. The motion always stops exactly at this point.</p> <p>The coordinates refer to the BASE coordinate system.</p>
	<i>End point</i>	<p>Type: POS, E6POS, FRAME</p> <p>If not all components of the end point are specified, the controller takes the values of the previous position for the missing components.</p> <p>The Status and Turn specifications for an end point of type POS or E6POS are disregarded.</p> <p>The coordinates refer to the BASE coordinate system.</p>
	<i>Circular angle</i>	<p>Unit: Degrees; without restriction</p> <p>(&gt;&gt;&gt; 9.9 "Circular angle" Page 306)</p>
	<i>Approximation (blending)</i>	<p><i>Approximation</i> causes the end point to be approximated. At the same time, this parameter defines the earliest point at which the approximate positioning can begin.</p> <p>(&gt;&gt;&gt; 11.6.7 "Approximation parameters for PTP, LIN CIRC and ..._REL" Page 378)</p>

**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
```

**11.6.4 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: ([">>>> 11.6.8 "REL motions for infinitely rotating axes"](#) Page 379)

**Syntax**

```
PTP_REL End point <Approximation> <#BASE | #TOOL>
```

**Explanation of the syntax**

Element	Description
<i>End point</i>	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.
<i>Approximation</i>	<i>Approximation</i> causes the end point to be approximated. At the same time, this parameter defines the earliest point at which the approximate positioning can begin.  ( <a href="#">"&gt;&gt;&gt;&gt; 11.6.7 "Approximation parameters for PTP, LIN CIRC and ..._REL"</a> Page 378)
#BASE , #TOOL	Only permissible if the end point was specified in Cartesian coordinates.  ■ #BASE (default): The coordinates of this end point refer to the coordinate system that belongs to the physical base. ■ #TOOL: The coordinates of this end point refer to the coordinate system that belongs to the physical tool.  \$IPO_MODE has no influence on the meaning of #BASE and #TOOL.

**Example 1**

Axis 2 is moved 30 degrees in a negative direction. None of the other axes moves.

```
PTP_REL {A2 -30}
```

**Example 2**

The robot moves 100 mm in the X direction and 200 mm in the negative Z direction from the current position. Y, A, B, C and S remain constant. T is calculated in relation to the shortest path.

```
PTP_REL {X 100,Z -200}
```

### 11.6.5 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: ([">>>> 11.6.8 "REL motions for infinitely rotating axes"](#) Page 379)

#### Syntax

`LIN_REL End point <Approximation> <#BASE | #TOOL>`

#### Explanation of the syntax

Element	Description
<i>End point</i>	<p>Type: POS, E6POS, FRAME</p> <p>The end point must be specified in Cartesian coordinates. The controller interprets the coordinates as relative to the current position.</p> <p>The coordinates can refer to the BASE or TOOL coordinate system.</p> <p>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.</p> <p>The Status and Turn specifications for an end point of type POS or E6POS are disregarded.</p>
<i>Approximation (blending)</i>	<p><i>Approximation</i> causes the end point to be approximated. At the same time, this parameter defines the earliest point at which the approximate positioning can begin.</p> <p>(<a href="#">"&gt;&gt;&gt;&gt; 11.6.7 "Approximation parameters for PTP, LIN CIRC and ..._REL"</a> Page 378)</p>
#BASE, #TOOL	<ul style="list-style-type: none"> <li>■ #BASE (default): The coordinates of this end point refer to the coordinate system that belongs to the physical base.</li> <li>■ #TOOL: The coordinates of this end point refer to the coordinate system that belongs to the physical tool.</li> </ul> <p>\$IPO_MODE has no influence on the meaning of #BASE and #TOOL.</p>



Information about \$APO is contained in the **System Variables** documentation.

#### Examples

##### Example 1:

The TCP moves 100 mm in the X direction and 200 mm in the negative Z direction from the current position in the BASE coordinate system. Y, A, B, C and S remain constant. T is determined by the motion.

```
LIN_REL {X 100,Z -200}
```

##### Example 2:

The TCP moves 100 mm from the current position in the negative X direction in the TOOL coordinate system. Y, Z, A, B, C and S remain constant. T is determined by the motion.

This example is suitable for moving the tool backwards against the tool direction. The precondition is that the tool direction has been calibrated along the X axis.

```
LIN_REL {X -100} #TOOL
```

### 11.6.6 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: ([">>>> 11.6.8 "REL motions for infinitely rotating axes"](#)  
Page 379)

#### Syntax

```
CIRC_REL Auxiliary point, End point<, CA Circular angle> <Approximation>  
<#BASE | #TOOL>
```

#### Explanation of the syntax

Element	Description
<i>Auxiliary point</i>	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.
<i>End point</i>	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.

Element	Description
<i>Circular angle</i>	Unit: Degrees; without restriction (>>> 9.9 "Circular angle" Page 306)
<i>Approximation (blending)</i>	<i>Approximation</i> causes the end point to be approximated. At the same time, this parameter defines the earliest point at which the approximate positioning can begin. (>>> 11.6.7 "Approximation parameters for PTP, LIN CIRC and ..._REL" Page 378)
#BASE, #TOOL	<ul style="list-style-type: none"> <li>■ #BASE (default): The coordinates of this end point refer to the coordinate system that belongs to the physical base.</li> <li>■ #TOOL: The coordinates of this end point refer to the coordinate system that belongs to the physical tool.</li> </ul> <p>\$IPO_MODE has no influence on the meaning of #BASE and #TOOL.</p>



Information about \$APO is contained in the **System Variables** documentation.

### Example

The end point of the circular motion is defined by a circular angle of 500°. The end point is approximated.

```
CIRC_REL {X 100,Y 3.2,Z -20},{Y 50},CA 500 C_VEL
```

### 11.6.7 Approximation parameters for PTP, LIN CIRC and ...\_REL

#### Parameters

Not every parameter can be used in every instruction.

Parameter	Description
C_PTP	Approximation starts, at the earliest, when half the distance between the start point and end point relative to the contour of the motion without approximation has been covered.
C_DIS	Approximation starts, at the earliest, when the distance to the end point falls below the value of \$APO.CDIS.
C_ORI	Approximation starts, at the earliest, when the dominant orientation angle falls below the value of \$APO.CORI.
C_VEL	Approximation starts, at the earliest, when the velocity in the deceleration phase to the end point falls below the value of \$APO.CVEL.



Information about \$APO is contained in the **System Variables** documentation.

#### PTP, PTP\_REL

The parameter must be C\_PTP or C\_DIS for **PTP-PTP** approximation. If a second parameter is specified, the robot controller ignores it.

Two parameters can be specified for **PTP-CP** approximation. Of the two parameters, the one resulting in the smaller approximate positioning radius in the given situation takes effect.

Possible combinations for PTP-CP approximation:

<b>1st parameter →</b>	<b>C_PTP</b>	<b>C_DIS</b>
<b>2nd parameter ↓</b>		
<b>(without)</b>	Possible	Possible
<b>C_DIS</b>	Possible	Not possible!
<b>C_VEL</b>	Possible	Possible
<b>C_ORI</b>	Possible	Possible

#### Example: PTP-CP approximation

```
PTP XP1 C_PTP C_DIS
LIN XP2
```

The robot controller calculates the approximate positioning radius which would result from each of the two parameters C\_PTP and C\_DIS under the current conditions (velocity, etc.). Only the smaller of the two radii then actually has an effect. It is the earliest limit at which approximate positioning can begin.

**LIN, CIRC,  
LIN\_REL,  
CIRC\_REL**

The parameter must be C\_DIS, C\_VEL or C\_ORI for **CP-CP** approximation. If a second parameter is specified, the robot controller ignores it.

Two parameters can be specified for **CP-PTP** approximation. Of the two parameters, the one resulting in the smaller approximate positioning radius in the given situation takes effect.

Possible combinations for CP-PTP approximation:

<b>1st parameter →</b>	<b>C_DIS</b>	<b>C_VEL</b>	<b>C_ORI</b>
<b>2nd parameter ↓</b>			
<b>(without)</b>	Possible	Possible	Possible
<b>C_PTP</b>	Possible	Possible	Possible
<b>C_DIS</b>	Possible	Possible	Possible

#### 11.6.8 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: <ul style="list-style-type: none"> <li>■ From “Start position minus 180°”</li> <li>■ To “Start position plus 180°”</li> </ul> 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:

- $450^\circ \pm (x * 360^\circ)$

The end position that meets these criteria is 90°.

$$450^\circ - (1 * 360^\circ) = 90^\circ$$

#### 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.

## 11.7 Motion programming: spline

### 11.7.1 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	(>>> 11.7.11 "System variables for WITH" Page 390)
Value	<p>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.</p> <p>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.</p> <p>(&gt;&gt;&gt; 11.12.3 "Constraints for functions in the trigger" Page 445)</p>
C_SPL	<ul style="list-style-type: none"> <li>■ With C_SPL: the end point is approximated. \$APO defines the earliest point at which the approximate positioning can begin.</li> <li>■ Without C_SPL: the motion stops exactly at the end point.</li> </ul>



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
```

### 11.7.2 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	(>>> 11.7.11 "System variables for WITH" Page 390)
Value	<p>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.</p> <p>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.</p> <p>(&gt;&gt;&gt; 11.12.3 "Constraints for functions in the trigger" Page 445)</p>
C_SPL	<ul style="list-style-type: none"> <li>■ With C_SPL: the end point is approximated. \$APO defines the earliest point at which the approximate positioning can begin.</li> <li>■ Without C_SPL: the motion stops exactly at the end point.</li> </ul>

#### 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
```

### 11.7.3 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	(>>> 11.7.11 "System variables for WITH" Page 390)
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 "Constraints for functions in the trigger" Page 445)
C_SPL	<ul style="list-style-type: none"> <li>■ 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.</li> <li>■ Without C_SPL: the motion stops exactly at the end point.</li> </ul>



In System Software 8.2 and earlier, the identifier for approximate positioning with spline was "C\_DIS". If programs based on 8.2 or older versions are used in higher versions of 8.x and contain C\_DIS, this can be retained and does not have to be changed to C\_SPL.

### 11.7.4 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	<p>Type: POS, E6POS, FRAME</p> <p>The coordinates refer to the BASE coordinate system.</p> <p>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.</p> <p>The end point does not adopt values from the auxiliary point, but from the position before.</p> <p>If no previous position is known to the robot controller, the missing components are taken from the current robot position.</p>
	Circular angle	<p>Unit: Degrees; without restriction</p> <p>(&gt;&gt;&gt; 9.9 "Circular angle" Page 306)</p>
	SysVar	<p>(&gt;&gt;&gt; 11.7.11 "System variables for WITH" Page 390)</p>
	Value	<p>Value assignment to the system variable.</p> <p>In the case of SCIRC segments: The assignment applies only for this segment.</p> <p>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.</p> <p>(&gt;&gt;&gt; 11.12.3 "Constraints for functions in the trigger" Page 445)</p>
	C_SPL	<ul style="list-style-type: none"> <li>■ With C_SPL: the end point is approximated. \$APO defines the earliest point at which the approximate positioning can begin.</li> <li>Only possible for individual motions, not for segments.</li> <li>■ Without C_SPL: the motion stops exactly at the end point.</li> </ul>



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
```

### 11.7.5 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	(>>> 11.7.11 "System variables for WITH" Page 390)
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 "Constraints for functions in the trigger" Page 445)

**Example**

```
SPL P4 WITH $ACC={CP 2.0, ORI1 200, ORI2 200}
```

**11.7.6 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	(>>> 11.7.11 "System variables for WITH" Page 390)

Element	Description
Value	<p>Value assignment to the system variable.</p> <p>In the case of SPTP segments: The assignment applies only for this segment.</p> <p>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.</p> <p>(&gt;&gt;&gt; 11.12.3 "Constraints for functions in the trigger" Page 445)</p>
C_SPL	<ul style="list-style-type: none"> <li>■ With C_SPL: the end point is approximated. \$APO defines the earliest point at which the approximate positioning can begin.</li> <li>Only possible for individual motions, not for segments.</li> <li>■ Without C_SPL: the motion stops exactly at the end point.</li> </ul>



In System Software 8.2 and earlier, the identifier for approximate positioning with spline was "C\_DIS". If programs based on 8.2 or older versions are used in higher versions of 8.x and contain C\_DIS, this can be retained and does not have to be changed to C\_SPL.

### 11.7.7 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: (>>> 11.6.8 "REL motions for infinitely rotating axes"  
Page 379)

#### Syntax

*SLIN\_REL End point <WITH SysVar1 = Value1 <, SysVar2 = Value2, ..., >>*  
*<C\_SPL><#BASE | #TOOL>*

#### Explanation of the syntax

Element	Description
<i>End point</i>	<p>Type: POS, E6POS, FRAME</p> <p>The point must be specified in Cartesian coordinates. The controller interprets the coordinates as relative to the end point of the previous motion.</p> <p>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.</p> <p>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!)</p>
<i>SysVar</i>	(>>> 11.7.11 "System variables for WITH" Page 390)

Element	Description
Value	<p>Value assignment to the system variable.</p> <p>In the case of segments: The assignment applies only for this segment.</p> <p>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.</p> <p>(&gt;&gt;&gt; 11.12.3 "Constraints for functions in the trigger" Page 445)</p>
C_SPL	<ul style="list-style-type: none"> <li>■ With C_SPL: the end point is approximated. \$APO defines the earliest point at which the approximate positioning can begin.</li> <li>Only possible for individual motions, not for segments.</li> <li>■ Without C_SPL: the motion stops exactly at the end point.</li> </ul>
#BASE , #TOOL	<ul style="list-style-type: none"> <li>■ #BASE (default): The coordinates of this end point refer to the coordinate system that belongs to the physical base.</li> <li>■ #TOOL: The coordinates of this end point refer to the coordinate system that belongs to the physical tool.</li> </ul> <p>\$IPO_MODE has no influence on the meaning of #BASE and #TOOL.</p>

**Example**

```

DECL E6POS P1 = {X 1500, Y -200, Z 2000, A 0, B 0, C 0, S 6, T27}

SPTP HOME
SLIN P1

SLIN_REL{X 0, Y 500, Z 0, A 0, B 0, C 0} WITH $BASE=$NULLFRAME #BASE
SLIN_REL{X 400} WITH $TOOL=$NULLFRAME C_SPL #TOOL
SLIN_REL{A 20}
SPTP_REL{A3 90} C_SPL
SPTP_REL{Z 50, B -30} WITH $VEL.AXIS[4]=90 C_SPL #TOOL
SPTP_REL{A1 100}

SPLINE
    SPL P1
    SPL_REL{Z -300, B50} #TOOL
ENDSPLINE
PTPSPLINE
    SPTP P1
    SPTP_REL{A1 -100, A5 -70}
ENDSPLINE

```

**11.7.8 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: (>>> 11.6.8 "REL motions for infinitely rotating axes"  
Page 379)

<b>Syntax</b>	SCIRC_REL <i>Auxiliary point, End point &lt;, CA Circular angle&gt; &lt;WITH SysVar1 = Value1 &lt;, SysVar2 = Value2 , ... &gt;&gt; &lt;C_SPL&gt;&lt;#BASE   #TOOL&gt;</i>																
<b>Explanation of the syntax</b>	<table border="1"> <thead> <tr> <th style="background-color: #cccccc;">Element</th><th style="background-color: #cccccc;">Description</th></tr> </thead> <tbody> <tr> <td><i>Auxiliary point</i></td><td>Type: POS, E6POS, FRAME</td></tr> <tr> <td><i>End point</i></td><td> <p>The point must be specified in Cartesian coordinates. The controller interprets the coordinates as relative to the end point of the previous motion.</p> <p>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.</p> <p>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!)</p> <p>At the auxiliary point, the orientation angles are also ignored.</p> <p>The auxiliary point cannot be approximated. The motion always stops exactly at this point.</p> </td></tr> <tr> <td><i>Circular angle</i></td><td>Unit: Degrees; without restriction  (&gt;&gt;&gt; 9.9 "Circular angle" Page 306)</td></tr> <tr> <td><i>SysVar</i></td><td>(&gt;&gt;&gt; 11.7.11 "System variables for WITH" Page 390)</td></tr> <tr> <td><i>Value</i></td><td> <p>Value assignment to the system variable.</p> <p>In the case of segments: The assignment applies only for this segment.</p> <p>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.</p> <p>(&gt;&gt;&gt; 11.12.3 "Constraints for functions in the trigger" Page 445)</p> </td></tr> <tr> <td><i>C_SPL</i></td><td> <ul style="list-style-type: none"> <li>■ With C_SPL: the end point is approximated. \$APO defines the earliest point at which the approximate positioning can begin.</li> <li>Only possible for individual motions, not for segments.</li> <li>■ Without C_SPL: the motion stops exactly at the end point.</li> </ul> </td></tr> <tr> <td><i>#BASE , #TOOL</i></td><td> <ul style="list-style-type: none"> <li>■ #BASE (default): The coordinates of this end point refer to the coordinate system that belongs to the physical base.</li> <li>■ #TOOL: The coordinates of this end point refer to the coordinate system that belongs to the physical tool.</li> </ul> <p>\$IPO_MODE has no influence on the meaning of #BASE and #TOOL.</p> </td></tr> </tbody> </table>	Element	Description	<i>Auxiliary point</i>	Type: POS, E6POS, FRAME	<i>End point</i>	<p>The point must be specified in Cartesian coordinates. The controller interprets the coordinates as relative to the end point of the previous motion.</p> <p>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.</p> <p>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!)</p> <p>At the auxiliary point, the orientation angles are also ignored.</p> <p>The auxiliary point cannot be approximated. The motion always stops exactly at this point.</p>	<i>Circular angle</i>	Unit: Degrees; without restriction  (>>> 9.9 "Circular angle" Page 306)	<i>SysVar</i>	(>>> 11.7.11 "System variables for WITH" Page 390)	<i>Value</i>	<p>Value assignment to the system variable.</p> <p>In the case of segments: The assignment applies only for this segment.</p> <p>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.</p> <p>(&gt;&gt;&gt; 11.12.3 "Constraints for functions in the trigger" Page 445)</p>	<i>C_SPL</i>	<ul style="list-style-type: none"> <li>■ With C_SPL: the end point is approximated. \$APO defines the earliest point at which the approximate positioning can begin.</li> <li>Only possible for individual motions, not for segments.</li> <li>■ Without C_SPL: the motion stops exactly at the end point.</li> </ul>	<i>#BASE , #TOOL</i>	<ul style="list-style-type: none"> <li>■ #BASE (default): The coordinates of this end point refer to the coordinate system that belongs to the physical base.</li> <li>■ #TOOL: The coordinates of this end point refer to the coordinate system that belongs to the physical tool.</li> </ul> <p>\$IPO_MODE has no influence on the meaning of #BASE and #TOOL.</p>
Element	Description																
<i>Auxiliary point</i>	Type: POS, E6POS, FRAME																
<i>End point</i>	<p>The point must be specified in Cartesian coordinates. The controller interprets the coordinates as relative to the end point of the previous motion.</p> <p>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.</p> <p>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!)</p> <p>At the auxiliary point, the orientation angles are also ignored.</p> <p>The auxiliary point cannot be approximated. The motion always stops exactly at this point.</p>																
<i>Circular angle</i>	Unit: Degrees; without restriction  (>>> 9.9 "Circular angle" Page 306)																
<i>SysVar</i>	(>>> 11.7.11 "System variables for WITH" Page 390)																
<i>Value</i>	<p>Value assignment to the system variable.</p> <p>In the case of segments: The assignment applies only for this segment.</p> <p>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.</p> <p>(&gt;&gt;&gt; 11.12.3 "Constraints for functions in the trigger" Page 445)</p>																
<i>C_SPL</i>	<ul style="list-style-type: none"> <li>■ With C_SPL: the end point is approximated. \$APO defines the earliest point at which the approximate positioning can begin.</li> <li>Only possible for individual motions, not for segments.</li> <li>■ Without C_SPL: the motion stops exactly at the end point.</li> </ul>																
<i>#BASE , #TOOL</i>	<ul style="list-style-type: none"> <li>■ #BASE (default): The coordinates of this end point refer to the coordinate system that belongs to the physical base.</li> <li>■ #TOOL: The coordinates of this end point refer to the coordinate system that belongs to the physical tool.</li> </ul> <p>\$IPO_MODE has no influence on the meaning of #BASE and #TOOL.</p>																

### 11.7.9 SPL\_REL: relative motion programming

<b>Description</b>	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: (>>> 11.6.8 "REL motions for infinitely rotating axes" Page 379)
<b>Syntax</b>	SPL_REL <i>End point &lt; WITH SysVar1 = Value1 &lt;, SysVar2 = Value2 , ... &gt;&gt;&gt; &lt;#BASE   #TOOL&gt;</i>

**Explanation of  
the syntax**

<b>Element</b>	<b>Description</b>
<i>End point</i>	<p>Type: POS, E6POS, FRAME</p> <p>The point must be specified in Cartesian coordinates. The controller interprets the coordinates as relative to the end point of the previous motion.</p> <p>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.</p> <p>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!)</p>
SysVar	(>>> 11.7.11 "System variables for WITH" Page 390)
Value	<p>Value assignment to the system variable.</p> <p>In the case of segments: The assignment applies only for this segment.</p> <p>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.</p> <p>(&gt;&gt;&gt; 11.12.3 "Constraints for functions in the trigger" Page 445)</p>
C_SPL	<ul style="list-style-type: none"> <li>■ With C_SPL: the end point is approximated. \$APO defines the earliest point at which the approximate positioning can begin.</li> <li>Only possible for individual motions, not for segments.</li> <li>■ Without C_SPL: the motion stops exactly at the end point.</li> </ul>
#BASE , #TOOL	<ul style="list-style-type: none"> <li>■ #BASE (default): The coordinates of this end point refer to the coordinate system that belongs to the physical base.</li> <li>■ #TOOL: The coordinates of this end point refer to the coordinate system that belongs to the physical tool.</li> </ul> <p>\$IPO_MODE has no influence on the meaning of #BASE and #TOOL.</p>

**Example**

(&gt;&gt;&gt; "Example" Page 387)

### 11.7.10 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: (>>> 11.6.8 "REL motions for infinitely rotating axes" Page 379)

**Syntax**

```
SPTP_REL End point <WITH SysVar1 = Value1 <, SysVar2 = Value2, ...>>
<C_SPL><#BASE | #TOOL>
```

Explanation of the syntax	Element	Description
	End point	<p>Type: AXIS, E6AXIS, POS, E6POS, FRAME</p> <p>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.</p> <p>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.</p> <p>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!)</p>
	SysVar	<p>(&gt;&gt;&gt; 11.7.11 "System variables for WITH" Page 390)</p>
	Value	<p>Value assignment to the system variable.</p> <p>In the case of SPTP segments: The assignment applies only for this segment.</p> <p>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.</p> <p>(&gt;&gt;&gt; 11.12.3 "Constraints for functions in the trigger" Page 445)</p>
	C_SPL	<ul style="list-style-type: none"> <li>■ With C_SPL: the end point is approximated. \$APO defines the earliest point at which the approximate positioning can begin.</li> <li>Only possible for individual motions, not for segments.</li> <li>■ Without C_SPL: the motion stops exactly at the end point.</li> </ul>
	#BASE, #TOOL	<p>Only permissible if the end point was specified in Cartesian coordinates.</p> <ul style="list-style-type: none"> <li>■ #BASE (default): The coordinates of this end point refer to the coordinate system that belongs to the physical base.</li> <li>■ #TOOL: The coordinates of this end point refer to the coordinate system that belongs to the physical tool.</li> </ul> <p>\$IPO_MODE has no influence on the meaning of #BASE and #TOOL.</p>

**Example**

(>>> "Example" Page 387)

**11.7.11 System variables for WITH**

<b>Spline block, individual spline motion</b>	For spline blocks and individual spline motions, it is possible to write to the following system variables using the WITH line:
	\$ACC
	\$ACC_AXIS
	\$ACC_EXTAX
	\$APO
	\$BASE
	\$CIRC_TYPE

\$COLLMON\_TOL\_PRO  
 \$DERI  
 \$ECO\_LEVEL  
 \$EXT\_IPO\_MODE  
 \$GEAR\_JERK  
 \$IPO\_MODE  
 \$JERK  
 \$JOINT\_OFFSET  
 \$LASER\_MODE  
 \$LOAD  
 \$LOAD\_A1, \$LOAD\_A2, \$LOAD\_A3  
 \$ORI\_TYPE  
 \$ROTSYS  
 \$SPL\_ORI\_JOINT\_AUTO  
 \$SPL\_VEL\_MODE  
 \$SPL\_VEL\_RESTR  
 \$SYNC\_ID  
 \$SYNC\_LIST  
 \$TENSION  
 \$TOOL  
 \$VEL  
 \$VEL\_AXIS  
 \$VEL\_EXTAX  
 Additionally for SCIRC: \$CIRC\_MODE

**Spline segment**

For spline segments, it is possible to write to the following system variables using the WITH line:

\$ACC  
 \$ACC\_AXIS  
 \$ACC\_EXTAX  
 \$CIRC\_TYPE  
 \$COLLMON\_TOL\_PRO  
 \$DERI  
 \$EX\_AX\_IGNORE  
 \$GEAR\_JERK  
 \$JERK  
 \$ORI\_TYPE  
 \$ROTSYS  
 \$SYNC\_ID  
 \$TENSION  
 \$VEL  
 \$VEL\_AXIS

\$VEL\_EXTAX  
Additionally for SCIRC: \$CIRC\_MODE

### 11.7.12 TIME\_BLOCK: programming a time block for spline

<b>Description</b>	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.
<b>Syntax</b>	<pre>SPLINE &lt;Spline segments...&gt; ... TIME_BLOCK START Spline segment &lt;Spline segments...&gt; ... &lt;&lt;TIME_BLOCK PART = Component_1&gt; ... Spline segment &lt;Spline segments...&gt; ... TIME_BLOCK PART = Component_N&gt; TIME_BLOCK END = Overall time &lt;Spline segments...&gt; ... ENDSPLINE</pre>
<b>Explanation of the syntax</b>	<p>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:</p> <ul style="list-style-type: none"><li>■ There is at least 1 spline segment between SPLINE and TIME_BLOCK START.</li><li>■ There is at least 1 spline segment between TIME_BLOCK END and END-SPLINE.</li></ul> <p>Advantages:</p> <ul style="list-style-type: none"><li>■ The programmed overall time is maintained exactly even in the case of approximate positioning.</li></ul>

- Segments before TIME\_BLOCK START make it possible to accelerate to the required velocity.

Element	Description
<i>Component</i>	<p>Type: INT or REAL; constant, variable or function</p> <p>Desired component of <i>Overall time</i> for the following distance:</p> <ul style="list-style-type: none"> <li>■ From the point before TIME_BLOCK PART=<i>Previous_component</i> to the point before TIME_BLOCK PART=<i>Component</i></li> <li>■ If <i>Previous_component</i> does not exist: From the point before TIME_BLOCK START to the point before TIME_BLOCK PART=<i>Component</i></li> </ul> <p>“Desired component” means: the components are maintained as accurately as possible by the robot controller. Generally, however, they are not maintained exactly.</p> <p>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 <i>Overall time</i>.</p> <p>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 <i>Overall time</i>. This allows the components to be used very flexibly and also changed.</p> <p>If components are assigned, there must always be a TIME_BLOCK PART directly before TIME_BLOCK END. There must be no segments in between.</p>
<i>Overall time</i>	<p>Type: INT or REAL; constant, variable or function; unit: s</p> <p>Time in which the following distance is traveled:</p> <ul style="list-style-type: none"> <li>■ From the point before TIME_BLOCK START to the point before TIME_BLOCK END</li> </ul> <p>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.</p>



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 "Constraints for functions in the trigger" Page 445)

## 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
in the time block	The spline block is executed as if there were no TIME_BLOCK statements present.
TIME_BLOCK END	
in the spline block after TIME_BLOCK END	

If the robot controller does not plan the time block, it generates the following message: *Time block ignored due to BCO run.*

#### \$PATHTIME

The data of the time-based spline can be read via the system variable \$PATHTIME. \$PATHTIME is filled with the data as soon as the robot controller has completed the planning of the spline block. The data are retained until the next spline block has been planned.

\$PATHTIME is a structure and consists of the following components:

Component	Description
REAL \$PATHTIME.TOTAL	Time actually required for the entire spline block (s)
REAL \$PATHTIME.SCHEDULED	Overall time planned for the time block (s)
REAL \$PATHTIME.PROGRAMMED	Overall time programmed for the time block (s)
INT \$PATHTIME.N_SECTIONS	Number N of TIME_BLOCK_PART lines
REAL \$PATHTIME.MAX_DEV	Maximum deviation of all TIME_BLOCK_PARTs between the programmed time and the planned time (%)
INT \$PATHTIME.MAX_DEV_SECTION	Number of the TIME_BLOCK_PART with the greatest deviation between the programmed time and the planned time

### 11.7.13 CONST\_VEL: programming a constant velocity range for spline motions

#### Description

CONST\_VEL is used to define constant velocity ranges. CONST\_VEL can only be used in CP spline blocks.

There must be at least 1 spline segment between CONST\_VEL END and ENDSPLINE.

A CP spline block can contain either 1 CONST\_VEL or 1 TIME\_BLOCK, but not both.



Basic information about the constant velocity range can be found here:

(>>> 10.4.6 "Constant velocity range in the CP spline block"

Page 340)

#### Syntax

CONST\_VEL START = Offset <ONSTART>

<Spline segments...>

...

CONST\_VEL END = Offset <ONSTART>

#### Explanation of the syntax

Element	Description
ONSTART	<p>Reference point of the statement</p> <ul style="list-style-type: none"> <li>■ With ONSTART: Start point</li> <li>■ Without ONSTART: End point</li> </ul> <p>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.</p> <p>(&gt;&gt;&gt; 11.12.2.2 "Reference point for homogenous approximate positioning" Page 443)</p>
Offset	<p>Type: INT or REAL; constant, variable or function; unit: mm</p> <p>The start of the range can be shifted in space by means of CONST_VEL START = Offset.</p> <p>The end of the range can be shifted in space by means of CONST_VEL END = Offset.</p> <ul style="list-style-type: none"> <li>■ Positive value: Offset towards the end of the motion</li> <li>■ Negative value: Offset towards the start of the motion</li> </ul> <p>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.</p> <p>(&gt;&gt;&gt; 10.4.6.2 "Maximum limits" Page 341)</p>



The value for Offset can be assigned using a function. The same restrictions apply as for the functions in the trigger.

(>>> 11.12.3 "Constraints for functions in the trigger" Page 445)

#### 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

```

### 11.7.13.1 System variables for CONST\_VEL

**\$STOP\_CONST\_VEL\_RED** If the maximum possible constant velocity in a constant velocity range is below the programmed velocity, the robot controller generates one of the following messages:

- In the CONST\_VEL END range, instead of \$VEL.CP={Setpoint \$VEL.CP} m/s only {Velocity reached} m/s because line {Line of the limiting segment} reached.
- In CONST\_VEL START instead of \$VEL.CP={Setpoint \$VEL.CP} m/s only {Velocity reached} m/s because line {Line of the limiting segment} reached.
- In CONST\_VEL END instead of \$VEL.CP={Setpoint \$VEL.CP} m/s only {Velocity reached} m/s because line {Line of the limiting segment} reached.

For operating modes T1/T2, it is possible to configure whether these are notification messages or acknowledgement messages. This is carried out using the system variable \$STOP\_CONST\_VEL\_RED.

Value	Description
FALSE (default)	Notification message
TRUE	<ul style="list-style-type: none"> <li>■ Operating mode T1 or T2: Acknowledgement message The robot stops. In the program, the block pointer indicates the spline segment that triggered the stop. Program execution cannot be resumed until the operator has acknowledged the message.</li> <li>■ Operating mode AUT or AUT EXT: Notification message</li> </ul>

\$STOP\_CONST\_VEL\_RED is initialized with FALSE in the case of a cold start or program selection, but not if a program is reset.

\$STOP\_CONST\_VEL\_RED can be modified via the variable correction function or KRL.

### \$CONST\_VEL

\$CONST\_VEL specifies the velocity (mm/s) in the constant velocity range for the CP spline block that is currently being planned. The value remains valid until a different spline block with constant velocity range is planned.

\$CONST\_VEL is write-protected. \$CONST\_VEL is invalid if one of the following motions is currently being planned:

- CP spline block without constant velocity range
- PTP spline block
- Individual spline motion
- PTP, LIN, CIRC

**\$CONST\_VEL\_C** \$CONST\_VEL\_C corresponds to \$CONST\_VEL, but with the difference that \$CONST\_VEL\_C refers to the CP spline block that is currently being executed.

Possible practical use:

The value of \$CONST\_VEL can be written to a user-specific local variable for each constant velocity range. This provides an overview of the constant velocities that can be reached. The application can then be programmed accordingly, e.g. the point in time at which a nozzle is to be opened, or similar.

### 11.7.14 STOP WHEN PATH: programming a conditional stop for spline

**Description** The operator can program a conditional stop with STOP WHEN PATH.



Further information about the conditional stop can be found in this documentation.  
(>>> 10.4.5 "Conditional stop" Page 337)

**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 END-SPLINE.

- 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	<p>Reference point of the statement</p> <ul style="list-style-type: none"><li>■ Without ONSTART: End point</li><li>■ With ONSTART: Start point</li></ul> <p>If the reference point is approximated, the same rules apply as for the PATH trigger.</p> <p>(&gt;&gt;&gt; 11.12.2.1 "Reference point for approximate positioning – overview" Page 442)</p>

Element	Description
Offset	<p>Type: INT or REAL; constant, variable or function; unit: mm</p> <p>The stop point can be shifted in space by means of <i>Offset</i>.</p> <ul style="list-style-type: none"> <li>■ Positive value: Shift towards the end of the motion</li> <li>■ Negative value: Shift towards the start of the motion</li> </ul> <p>The point to which the offset refers depends on whether ONSTART is set or not. If no offset is desired, <i>Offset=0</i> must be programmed.</p> <p>There are limits to the distance the stop point can be offset. The same limits apply as for the PATH trigger. (<a href="#">&gt;&gt;&gt; "Max. offset" Page 440</a>)</p>
Condition	<p>Type: BOOL</p> <p>Stop condition. The following are permitted:</p> <ul style="list-style-type: none"> <li>■ a global Boolean variable</li> <li>■ a signal name</li> <li>■ a comparison</li> <li>■ a simple logic operation: NOT, OR, AND or EXOR</li> </ul>



The value for *Offset* can be assigned using a function. The same restrictions apply as for the functions in the trigger.  
[\(>>> 11.12.3 "Constraints for functions in the trigger" Page 445\)](#)

### 11.7.15 \$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	<ul style="list-style-type: none"><li>■ <b>Bit n = 1:</b> Taught/programmed position of the external axis is ignored.</li><li>■ <b>Bit n = 0:</b> Taught/programmed position of the external axis is taken into consideration.</li></ul>

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.

## 11.8 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: exiting a loop

**Description** Exit from a loop. The program is then continued after the loop. EXIT may be used in any loop.

**Syntax** EXIT

**Example** The loop is exited when \$IN[1] is set to TRUE. The program is then continued after ENDLOOP.

```
DEF EXIT_PROG()
PTP HOME
LOOP
    PTP POS_1
    PTP POS_2
    IF $IN[1] == TRUE THEN
        EXIT
    ENDIF
    CIRC HELP_1, POS_3
    PTP POS_4
ENDLOOP
PTP HOME
END
```

### 11.8.3 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. In the case of nested loops, the outer loop is executed completely first. The inner loop is then executed completely.

**Syntax**

```
FOR Counter = Start value TO End value <STEP Increment>
<Statements>
ENDFOR
```

**Explanation of the syntax**

Element	Description
Counter	Type: INT  Variable that counts the number of times the loop has been executed. The preset value is <i>Start</i> . The variable must first be declared.  The value of <i>Counter</i> can be used in statements inside and outside of the loop. Once the loop has been exited, <i>Counter</i> retains its most recent value.

Element	Description
<i>Start; End</i>	Type: INT  <i>Counter</i> must be preset to the value <i>Start</i> . Each time the loop is executed, the value of <i>Counter</i> is automatically increased by the increment. If the value exceeds or falls below the <i>End</i> value, the loop is terminated.
<i>Increment</i>	Type: INT  Value by which <i>Counter</i> is changed every time the loop is executed. The value may be negative. Default value: 1. <ul style="list-style-type: none"><li>■ Positive value: the loop is ended if <i>Counter</i> is greater than <i>End</i>.</li><li>■ Negative value: the loop is ended if <i>Counter</i> is less than <i>End</i>.</li></ul> 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
<i>Label</i>	Position to which a jump is made. At the destination position, <i>Label</i> 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

<b>Description</b>	<p>Stops the program. The last motion instruction to be executed will, however, be completed.</p> <p>Execution of the program can only be resumed using the Start key. The next instruction after HALT is then executed.</p> <p>In an interrupt program, program execution is only stopped after the advance run has been completely executed.</p>
<b>Syntax</b>	HALT

### 11.8.6 IF ... THEN ... ENDIF: programming a conditional branch

<b>Description</b>	<p>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.</p> <p>The ELSE block may be omitted. If the condition is not satisfied, the program is then continued at the position immediately after ENDIF.</p> <p>There is no limit on the number of statements contained in the statement blocks. Several IF statements can be nested in each other.</p>
<b>Syntax</b>	<pre> IF Condition THEN     Statements &lt;ELSE     Statements&gt; ENDIF </pre>

#### Explanation of the syntax

Element	Description
<i>Condition</i>	<p>Type: BOOL</p> <p>Possible:</p> <ul style="list-style-type: none"> <li>■ Variable of type BOOL</li> <li>■ Function of type BOOL</li> <li>■ Logic operation, e.g. a comparison, with a result of type BOOL</li> </ul>

#### 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. In the case of nested loops, the outer loop is executed completely first. The inner loop is then executed completely.

**Syntax** LOOP

*Statements*

ENDLOOP

**Example** The loop is executed until input \$IN[30] is set to true.

```
LOOP  
    LIN P_1  
    LIN P_2  
    IF $IN[30]==TRUE THEN  
        EXIT  
    ENDIF  
ENDLOOP
```

### 11.8.8 ON\_ERROR\_PROCEED: suppressing a runtime error message

**Description** ON\_ERROR\_PROCEED can be used to suppress a runtime error message triggered by the following program line. The robot controller skips the statement that triggers the error and fills the system variable \$ERR with information about the error.

Messages about internal errors or system errors cannot be suppressed.

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.

**\$ERR,**  
**ERR\_RAISE()** \$ERR and ERR\_RAISE() are important tools when working with ON\_ERROR\_PROCEED.

The function ERR\_RAISE() can subsequently generate a suppressed runtime error message. It can only process the system variable \$ERR or a variable derived from \$ERR as an OUT parameter.

**Limitations** ON\_ERROR\_PROCEED has no effect on motion statements:

SPLINE/ENDSPLINE; PTP\_SPLINE/ENDSPLINE; PTP; LIN; CIRC; PTP\_REL; LIN\_REL; CIRC\_REL; ASYPTP; ASYSTOP; ASYCONT; ASYCANCEL; MOVE\_EMI

ON\_ERROR\_PROCEED has no effect on the following control structures:

FOR/ENDFOR; GOTO; IF/ELSE/ENDIF; LOOP/ENDLOOP; REPEAT/UNTIL; SKIP/ENDSKIP; SWITCH/CASE/DEFAULT/ENDSWITCH; WHILE/ENDWHILE

<b>Syntax</b>	ON_ERROR_PROCEED
<b>Examples</b>	(>>> 11.8.8.2 "Examples of \$ERR, ON_ERROR_PROCEED and ERR_RAISE()" Page 406)

**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.

**11.8.8.1 \$ERR**

<b>Description</b>	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 fault service function.  The variable is write-protected and can only be read.  \$ERR exists separately for the robot and submit interpreters. Each interpreter can only access its own variable. \$ERR does not exist for the command interpreter.  Each subprogram level has its own representation of \$ERR. In this way, the information from one level does not overwrite the information from different levels and information can be read from different levels simultaneously.  ON_ERROR_PROCEED implicitly deletes the information from \$ERR in the current interpreter and at the current level.
(>>> 11.8.8.2 "Examples of \$ERR, ON_ERROR_PROCEED and ERR_RAISE()" Page 406)	

<b>Syntax</b>	\$ERR=Information				
<b>Explanation of the syntax</b>	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="background-color: #cccccc;">Element</th> <th style="background-color: #cccccc;">Description</th> </tr> </thead> <tbody> <tr> <td>Information</td> <td>Type: Error_T  List with information about the program currently being executed</td> </tr> </tbody> </table>	Element	Description	Information	Type: Error_T  List with information about the program currently being executed
Element	Description				
Information	Type: Error_T  List with information about the program currently being executed				

**Error\_T**

```
STRUCT Error_T INT number, PROG_INT_E interpreter,
INT_TYP_E int_type, INT int_prio, line_nr, CHAR mo-
dule[24], up_name[24], TRIGGER_UP_TYPE trigger_type
```

<b>Element</b>	<b>Description</b>
number	Only in the event of a runtime error: Message number If no error has occurred, the value zero is displayed.
interpreter	Current interpreter  The component can take the following values, regardless of whether the robot controller is operated in Single Submit mode or in Multi-Submit mode.  Single Submit mode: <ul style="list-style-type: none"> <li>■ #R_INT: Robot interpreter</li> <li>■ #S_INT: Submit interpreter</li> </ul> Multi-Submit mode: <ul style="list-style-type: none"> <li>■ #R_INT: Robot interpreter</li> <li>■ #S_INT: System submit interpreter</li> <li>■ #EXT_S_INT1: Extended submit interpreter 1</li> <li>■ #EXT_S_INT2: Extended submit interpreter 2</li> <li>■ Etc.</li> </ul>
int_type	Current program type and interrupt state <ul style="list-style-type: none"> <li>■ #I_NORMAL: The program is not an interrupt program.</li> <li>■ #I_INTERRUPT: The program is an interrupt program.</li> <li>■ #I_STOP_INTERRUPT: Interrupt by means of \$STOPMESS (error stop)</li> </ul>
int_prio	Priority of the interrupt
line_nr	Only in the event of a runtime error: Number of the line that triggered the error  <b>Note:</b> 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
up_name[]	Name of the current subprogram
trigger_type	Context in which the trigger belonging to a subprogram was triggered <ul style="list-style-type: none"> <li>■ #TRG_NONE: The subprogram is not a trigger subprogram.</li> <li>■ #TRG_REGULAR: The trigger subprogram was switched during forward motion.</li> <li>■ #TRG_BACKWARD: The trigger subprogram was switched during backward motion.</li> <li>■ #TRG_RESTART: The trigger subprogram was switched on switching back to forward motion.</li> <li>■ #TRG_REPLY: The trigger subprogram was switched repeatedly after backward motion.</li> </ul> <b>Note:</b> This component is available in System Software 8.3 or higher.

### 11.8.8.2 Examples of \$ERR, ON\_ERROR\_PROCEED and ERR\_RAISE()

#### Example 1

If you do not wish to suppress all possible runtime error messages, but only specific ones, this distinction can be made using SWITCH ... ENDSWITCH. In this example, only message 1422 is suppressed. Any other runtime error messages would be 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 triggers the message 1422 {\$variable} value invalid (unless the program is called by an interrupt). ON_ERROR_PROCEED in the preceding line suppresses the error message.
6 ... 12	Differentiation dependent on \$ERR.NUMBER
7	If no error occurred in line 5, \$ERR.NUMBER==0. In this case, no action is required.
8, 9	If message 1422 has been triggered, \$ERR.NUMBER==1422. If required, a fault service function can be programmed.
10, 11	If a message other than 1422 was triggered, this message is now (subsequently) generated via ERR_RAISE.

#### Example 2

This example 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

```

<b>Line</b>	<b>Description</b>
4, 5	<p>Line 5 triggers the message 1422 <i>{\$variable} value invalid</i> because myVar is not initialized and can thus not be transferred to a subprogram.</p> <p>ON_ERROR_PROCEED in the preceding line suppresses the error message.</p>
6	<p>If \$ERR is read here using the variable correction function, the following components have the following values:</p> <p>\$ERR.number == 1422  \$ERR.line_nr == 15  \$ERR.module[] == "MYMAINPROG"  \$ERR.up_name[] == "MYMAINPROG"</p>
12	<p>If \$ERR is read here in the subprogram using the variable correction function, the following components have the following values:</p> <p>\$ERR.number == 0  \$ERR.line_nr == 0  \$ERR.module[] == "MYMAINPROG"  \$ERR.up_name[] == "MYSUBPROG"</p> <p>This clearly indicates that \$ERR always has the information from the current level (from the subprogram MySubProg in this case). The information from MyMainProg, on the other hand, is unknown.</p>

**Example 3**

This example also 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

```

<b>Line</b>	<b>Description</b>
3, 4	<p>Line 4 triggers the message 1444 <i>Array index inadmissible</i>.  ON_ERROR_PROCEED in the preceding line suppresses the error message.</p>
5, 7	The contents of \$ERR are transferred to a subprogram twice: once as an IN parameter and once as an OUT parameter.

Line	Description
9, 10	Line 10 triggers the message 1451 <i>Division by 0</i> . ON_ERROR_PROCEED in the preceding line suppresses the error 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 <i>{(Variable name)} invalid argument</i> . ERR_RAISE can only process \$ERR or an OUT variable derived from \$ERR.

**Example 4**

\$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 ()
...
mySUB (55)
...
END
```

Submit program:

```
DEF SPS ()
...
LOOP
    mySUB (33)
    ...
ENDLOOP
...
END
```

Subprogram:

```
GLOBAL DEF mySUB (par:IN)
INT par
INI
IF ($ERR.INTERPRETER==#R_INT) THEN
    $OUT_C[par] = TRUE
ELSE
    $OUT[par] = TRUE
ENDIF
...
END
```

### 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. In the case of nested loops, the outer loop is executed completely first. The inner loop is then executed completely.

**Syntax**

REPEAT

*Statements*UNTIL *Termination condition***Explanation of the syntax**

Element	Description
<i>Break condition</i>	Type: BOOL  Possible: <ul style="list-style-type: none"><li>■ Variable of type BOOL</li><li>■ Function of type BOOL</li><li>■ Logic operation, e.g. a comparison, with a result of type BOOL</li></ul>

**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**

Selects one of several possible statement blocks, according to a selection criterion. Every statement block has at least one identifier. The block whose identifier matches the selection criterion is selected.

Once the block has been executed, the program is resumed after END-SWITCH.

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 >*

&lt;DEFAULT

*Default statement block>*

ENDSWITCH

DEFAULT may only occur once in a SWITCH statement.

### Explanation of the syntax

Element	Description
<i>Selection criterion</i>	Type: INT, CHAR, ENUM  This can be a variable, a function call or an expression of the specified data type.
<i>Identifier</i>	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 UP\_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
<i>Condition</i>	<p>Type: BOOL</p> <p>Condition, the fulfillment of which allows program execution to be resumed.</p> <ul style="list-style-type: none"> <li>■ If the condition is FALSE, program execution is stopped until the condition is TRUE.</li> <li>■ If the condition is already TRUE when WAIT is called, program execution is not halted.</li> </ul>

**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
<i>Wait time</i>	<p>Type: INT, REAL</p> <p>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.</p>

**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. In the case of nested loops, the outer loop is executed completely first. The inner loop is then executed completely.

**Syntax**

```
WHILE Repetition condition
```

*Statement block*

```
ENDWHILE
```

**Explanation of  
the syntax**

Element	Description
<i>Repetition condition</i>	Type: BOOL  Possible: <ul style="list-style-type: none"> <li>■ Variable of type BOOL</li> <li>■ Function of type BOOL</li> <li>■ Logic operation, e.g. a comparison, with a result of type BOOL</li> </ul>

**Example 1**

The loop is executed 99 times. After execution of the loop, *w* has the value 100.

```
W=1
WHILE W<100
    W=W+1
ENDWHILE
```

**Example 2**

The loop is executed as long as \$IN[1] is true.

```
WHILE $IN[1]==TRUE
    W=W+1
ENDWHILE
```

## 11.9 Inputs/outputs

### 11.9.1 ANIN: cyclically reading an analog input

**Description**

Cyclical reading (every 12 ms) of an analog input. ANIN triggers an advance run stop.

The robot controller has 32 analog inputs (\$ANIN[1] ... \$ANIN[32]).

- A maximum of three ANIN ON statements can be used at the same time.
- A maximum of two ANIN ON statements can use the same variable *Value* or access the same analog input.
- All of the variables used in an ANIN statement must be declared in data lists (locally or in \$CONFIG.DAT).

\$ANIN[...] indicates the input voltage, adapted to the range between -1.0 and +1.0. The actual voltage depends on the settings of the analog module.

**Syntax**

Starting cyclical reading:

```
ANIN ON Value = Factor * Signal name * <±Offset>
```

Ending cyclical reading:

```
ANIN OFF Signal name
```

**Explanation of  
the syntax**

Element	Description
<i>Value</i>	Type: REAL  The result of the cyclical reading is stored in <i>Value</i> . <i>Value</i> can be a variable or a signal name for an output.
<i>Factor</i>	Type: REAL  Any factor. It can be a constant, variable or signal name.

Element	Description
<i>Signal name</i>	Type: REAL  Specifies the analog input. <i>Signal name</i> must first have been declared with SIGNAL . It is not possible to specify the analog input \$ANIN[x] directly instead of the signal name.
<i>Offset</i>	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: 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
<i>Signal name</i>	Type: REAL  Specifies the analog output. <i>Signal name</i> must first have been declared with SIGNAL . It is not possible to specify the analog output \$ANOUT[x] directly instead of the signal name.
<i>Factor</i>	Type: REAL  Any factor. It can be a constant, variable or signal name.
<i>Control element</i>	Type: REAL  It can be a constant, variable or signal name.

Element	Description
Offset	Type: REAL It can be a constant, variable or signal name.
Time	Type: REAL Unit: seconds. By using the keyword DELAY and entering a positive or negative amount of time, the output signal can be delayed (+) or set early (-).
Minimum value, Maximum value	Type: REAL Minimum and/or maximum value to be present at the output. The actual value does not fall below/exceed these values, even if the calculated values fall outside this range. Permissible values: -1.0 to +1.0 It can be a constant, variable, structure component or array element. The minimum value must always be less than the maximum value. The sequence of the keywords MINIMUM and MAXIMUM must be observed.

**Example**

In this example, the output \$ANOUT[5] controls the adhesive output.

A freely selected name, in this case GLUE, is assigned to the analog output in the declaration section. The amount of adhesive is to be dependent on the current path velocity (= system variable \$VEL\_ACT). Furthermore, the output signal is to be generated 0.5 seconds early. The minimum voltage is to be 3 V. (The voltage of the module used ranges from +10 V to -10 V.)

```
SIGNAL GLUE $ANOUT[5]
...
ANOUT ON GLUE = 0.5 * $VEL_ACT DELAY=-0.5 MINIMUM=0.30
```

The cyclical analog output is ended by using ANOUT OFF:

```
ANOUT OFF GLUE
```

**11.9.3 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**

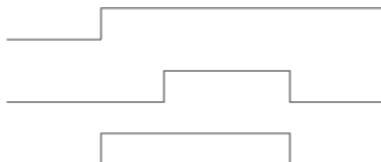
<b>Element</b>	<b>Description</b>
<i>Signal</i>	Type: BOOL  Output to which the pulse is to be fed. The following are permitted: <ul style="list-style-type: none"><li>■ OUT[No]</li><li>■ Signal variable</li></ul>
<i>Level</i>	Type: BOOL  Logic expression: <ul style="list-style-type: none"><li>■ TRUE represents a positive pulse (high).</li><li>■ FALSE represents a negative pulse (low).</li></ul>
<i>Pulse dura- tion</i>	Type: REAL  Range of values: 0.1 to 3.0 seconds. Pulse durations outside this range trigger a program stop.  Pulse interval: 0.1 seconds, i.e. the pulse duration is rounded up or down. The PULSE statement is executed in the controller at the low-priority clock rate. This results in a tolerance in the order of the pulse interval (0.1 seconds). The time deviation is about 1% - 2% on average. The deviation is about 13% for very short pulses.

**\$OUT+PULSE**

If an output is already set before the pulse, it will be reset by the falling edge of the pulse.

```
$OUT[50] = TRUE
PULSE($OUT[50], TRUE, 0.5)
```

Actual pulse characteristic at output 50:

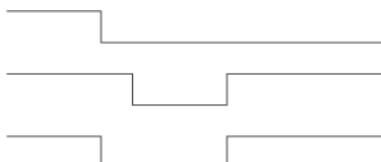


**Fig. 11-1: \$OUT+PULSE, example 1**

If a negative pulse is applied to an output that is set to Low, the output remains Low until the end of the pulse and is then set to High:

```
$OUT[50] = FALSE
PULSE($OUT[50], FALSE, 0.5)
```

Actual pulse characteristic at output 50:



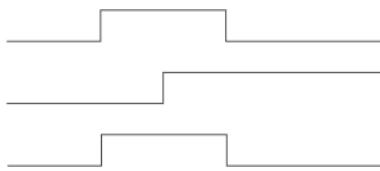
**Fig. 11-2: \$OUT+PULSE, example 2**

**PULSE+\$OUT**

If the same output is set during the pulse duration, it will be reset by the falling edge of the pulse.

```
PULSE($OUT[50], TRUE, 0.5)
$OUT[50] = TRUE
```

Actual pulse characteristic at output 50:

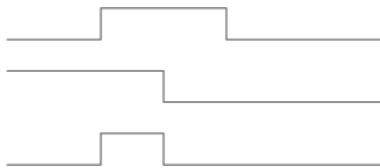


**Fig. 11-3: PULSE+\$OUT, example 1**

If the output is reset during the pulse duration, the pulse duration is reduced accordingly:

```
PULSE ($OUT[50] , TRUE , 0 . 5)
$OUT[50] = FALSE
```

Actual pulse characteristic at output 50:

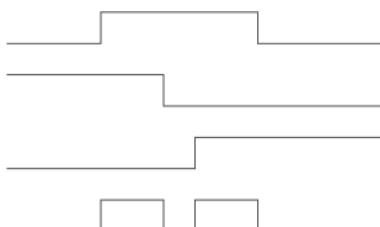


**Fig. 11-4: PULSE+\$OUT, example 2**

If an output is set to FALSE during a pulse and then back to TRUE, the pulse is interrupted and then resumed when the output is set to TRUE. The overall duration from the first rising edge to the last falling edge (i.e. including the duration of the interruption) corresponds to the duration specified in the PULSE statement.

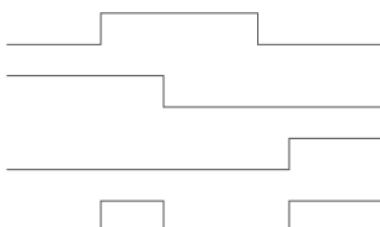
```
PULSE ($OUT[50] , TRUE , 0 . 8)
$OUT[50] =FALSE
$OUT[50] =TRUE
```

Actual pulse characteristic at output 50:



**Fig. 11-5: PULSE+\$OUT, example 3**

The actual pulse characteristic is only specified as above if \$OUT[x]=TRUE is set during the pulse. If \$OUT[x]=TRUE is not set until after the pulse (see line 3), then the actual pulse characteristic is as follows (line 4):



**Fig. 11-6: PULSE+\$OUT, example 4**

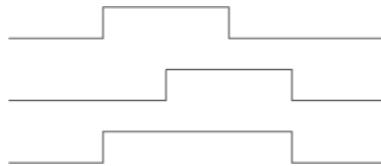
#### PULSE+PULSE

If several PULSE statements overlap, it is always the last PULSE statement that determines the end of the overall pulse duration.

If a pulse is activated again before the falling edge, the duration of the second pulse starts at this moment. The overall pulse duration is thus shorter than the sum of the values of the first and second pulses:

```
PULSE ($OUT[50], TRUE, 0.5)  
PULSE ($OUT[50], TRUE, 0.5)
```

Actual pulse characteristic at output 50:

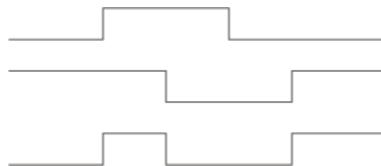


**Fig. 11-7: PULSE+PULSE, example 1**

If, during the pulse duration of a positive pulse, a negative pulse is sent to the same output, only the second pulse is taken into consideration from this moment onwards:

```
PULSE ($OUT[50], TRUE, 0.5)  
PULSE ($OUT[50], FALSE, 0.5)
```

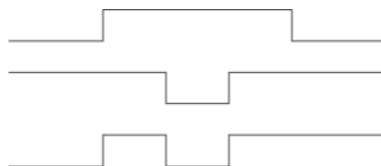
Actual pulse characteristic at output 50:



**Fig. 11-8: PULSE+PULSE, example 2**

```
PULSE ($OUT[50], TRUE, 3.0)  
PULSE ($OUT[50], FALSE, 1.0)
```

Actual pulse characteristic at output 50:



**Fig. 11-9: PULSE+PULSE, example 3**

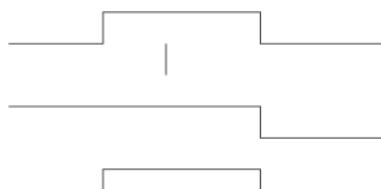
#### **PULSE+END**

If a pulse is programmed before the END statement, the duration of program execution is increased accordingly.

```
PULSE ($OUT[50], TRUE, 0.8)  
END
```

Program active

Actual pulse characteristic at output 50:



**Fig. 11-10: PULSE+END, example**

**PULSE+RESET/CANCEL** If program execution is reset (RESET) or aborted (CANCEL) while a pulse is active, the pulse is immediately reset:

```
PULSE ($OUT[50], TRUE, 0.8)
RESET OR CANCEL
```

Actual pulse characteristic at output 50:

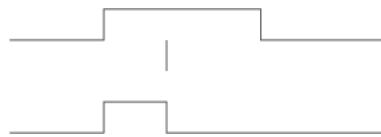


Fig. 11-11: PULSE+RESET, example

#### 11.9.4 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. (>>> 11.4.4 "Areas of validity" Page 367)
Signal name	Any name
Signal variable	Predefined signal variable. The following types are available: <ul style="list-style-type: none"> <li>■ \$IN[x]</li> <li>■ \$OUT[x]</li> <li>■ \$ANIN[x]</li> <li>■ \$ANOUT[x]</li> </ul>
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.

Element	Description
System signal name	Signal name predefined in the system, e.g. \$T1.
FALSE	<p>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.</p> <p>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.</p>

**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 421)
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 420)

**Example**

(>>> 11.10.4 "RETURN: jump back to the calling program" Page 420)

### 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
<i>Function value</i>	Type: The data type of <i>Function_Value</i> must match the data type of the function.  <i>Function_Value</i> 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 425)

**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.
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. <b>Note:</b> 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.
<b>Note:</b> 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

```

<b>Line</b>	<b>Description</b>
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:

<b>Type in the main program</b>	<b>Type in the subprogram</b>	<b>Effect</b>
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 ... : declaring an interrupt

<b>Description</b>	<p>In the case of a defined event, e.g. an input, the controller interrupts the current program and executes a defined subprogram. The event and the subprogram are defined by INTERRUPT ... DECL ... WHEN ... DO ....</p> <p>Once the subprogram has been executed, the interrupted program is resumed at the point at which it was interrupted. Exception: RESUME.</p> <p>A subprogram called by an interrupt is called an interrupt program.</p> <p>A maximum of 32 interrupts may be declared simultaneously. An interrupt declaration may be overwritten by another at any time.</p> <p>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.</p>
--------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------



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 428)



Interrupt programs must not contain any spline motions.

## Syntax

```
<GLOBAL> INTERRUPT <WITH BRAKE <F | FF>> DECL Prio WHEN Event
DO Subprogram
```

## Explanation of the syntax

Element	Description
GLOBAL	An interrupt is only recognized at, or below, the level in which it is declared. In other words, an interrupt declared in a subprogram is not recognized in the main program (and cannot be activated there). If an interrupt is also to be recognized at higher levels, the declaration must be preceded by the keyword GLOBAL.
WITH BRAKE <F   FF>	When the interrupt is detected, a BRAKE statement is executed. (>>> 11.11.4 "BRAKE: stopping the robot from an interrupt program" Page 432)
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.  <b>Note:</b> Priorities 3 and 40 to 80 are reserved for use by the system. They must not be used by the user because this would cause system-internal interrupts to be overwritten and result in errors.
Event	Type: BOOL  Event that is to trigger the interrupt. Structure components are impermissible. The following are permitted: <ul style="list-style-type: none"> <li>■ Global Boolean variables</li> <li>■ Signal names</li> <li>■ Comparisons</li> <li>■ Simple logic operations: NOT, OR, AND or EXOR</li> </ul>
Subprogram	The name of the interrupt program to be executed. Runtime variables must not be transferred to the interrupt program as parameters.

## Example 1

Declaration of an interrupt with priority 23 that calls the subprogram UP1 if \$IN[12] is true. The parameters 20 and VALUE are transferred to the subprogram.

```
INTERRUPT DECL 23 WHEN $IN[12]==TRUE DO UP1(20,VALUE)
```

**Example 2**

Two objects, the positions of which are detected by two sensors connected to inputs 6 and 7, are located on a programmed path. The robot is to be moved subsequently to these two positions.

For this purpose, the two detected positions are saved as points P\_1 and P\_2. These points are then addressed in the second section of the main program.

If the robot controller detects an event defined by means of INTERRUPT ... DECL ... WHEN ... DO ..., it always saves the current robot position in the system variables \$AXIS\_INT (axis-specific) and \$POS\_INT (Cartesian).

Main program:

```
DEF PROG()
...
INTERRUPT DECL 10 WHEN $IN[6]==TRUE DO UP1()
INTERRUPT DECL 20 WHEN $IN[7]==TRUE DO UP2()
...
INTERRUPT ON
LIN START
LIN END
INTERRUPT OFF
LIN P_1
LIN P_2
...
END
```

Local interrupt program 1:

```
DEF UP1()
P_1=$POS_INT
END
```

Local interrupt program 2:

```
DEF UP2()
P_2=$POS_INT
END
```

**Example 3**

If a collision is detected, the robot must be stopped quickly and reliably. For this purpose, an interrupt is declared with BRAKE F.

If the positive torque in axis A1 exceeds 1500 Nm, the interrupt immediately initiates a BRAKE F and calls the subprogram STOP\_FAST(). When the robot is stationary, a return motion is carried out to the Cartesian position \$POS\_INT, offset by -10 mm in tool direction X. \$POS\_INT is the position at which the interrupt was triggered.

```
DEF PROG()
...
INTERRUPT WITH BRAKE F DECL 25 WHEN $TORQUE_AXIS_ACT[1]>1500 DO
STOP_FAST()
INTERRUPT ON 25
...
END
...
DEF STOP_FAST()
BRAKE F
PTP $POS_INT:{x -10}
...
END
```

### 11.11.2 INTERRUPT ON/OFF: activating or deactivating an interrupt

<b>Description</b>	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.						
<b>Syntax</b>	<code>INTERRUPT Action &lt;Number&gt;</code>						
<b>Explanation of the syntax</b>	<table border="1"> <thead> <tr> <th>Element</th><th>Description</th></tr> </thead> <tbody> <tr> <td><i>Action</i></td><td> <ul style="list-style-type: none"> <li>■ <b>ON:</b> Activates an interrupt.</li> <li>■ <b>OFF:</b> Deactivates an interrupt.</li> </ul> </td></tr> <tr> <td><i>Number</i></td><td> <p>Type: INT</p> <p>Number (= priority) of the interrupt to which the <i>Action</i> is to refer.</p> <p><i>Number</i> can be omitted. In this case, ON or OFF refers to all declared interrupts.</p> </td></tr> </tbody> </table>	Element	Description	<i>Action</i>	<ul style="list-style-type: none"> <li>■ <b>ON:</b> Activates an interrupt.</li> <li>■ <b>OFF:</b> Deactivates an interrupt.</li> </ul>	<i>Number</i>	<p>Type: INT</p> <p>Number (= priority) of the interrupt to which the <i>Action</i> is to refer.</p> <p><i>Number</i> can be omitted. In this case, ON or OFF refers to all declared interrupts.</p>
Element	Description						
<i>Action</i>	<ul style="list-style-type: none"> <li>■ <b>ON:</b> Activates an interrupt.</li> <li>■ <b>OFF:</b> Deactivates an interrupt.</li> </ul>						
<i>Number</i>	<p>Type: INT</p> <p>Number (= priority) of the interrupt to which the <i>Action</i> is to refer.</p> <p><i>Number</i> can be omitted. In this case, ON or OFF refers to all declared interrupts.</p>						



Up to 16 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 16 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.

<b>Example 1</b>	The interrupt with priority 2 is activated. (The interrupt must already be declared.)
	<pre>INTERRUPT ON 2</pre>

<b>Example 2</b>	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).
	<pre>DEF PROG() ... INTERRUPT DECL 1 WHEN \$STOPMESS DO STOP_PROG() LIN P_1 INTERRUPT ON LIN P_2</pre>

```
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.

**Response** 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.

- 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 (e.g. an input was set to TRUE and then FALSE again), the interrupt responds regardless. The decisive factor is that the event has occurred (at least) once.

(>>> "Example 1" Page 430)

(>>> "Example 2" Page 430)

**Abnormal response** Under certain circumstances, an interrupt that has been re-enabled may not retroactively respond to an event. The decisive factor here is the flag for the events.

1. For all active interrupts (irrespective of whether they are disabled or not), the robot controller checks in the interpolation cycle whether a defined event has occurred.
2. If an event has occurred, the robot controller sets a flag.

The flag only exists once for all interrupts. It only recognizes TRUE and FALSE. Which event has occurred, and whether just one event or multiple events, is not taken into consideration. (This information is available to the interpreter elsewhere, however.)

3. For the interpreter, the set flag is the signal to execute relevant interrupt. If at least one interrupt is executed, the interpreter resets the flag.

Therefore, if a flag has been set for a disabled interrupt and a different, active interrupt is then executed, the flag is reset during execution of the interrupt. If the disabled interrupt is enabled, it can no longer retroactively respond to the event due to the missing flag.

(>>> "Example 3" Page 431)

(>>> "Example 4" Page 432)



Internal system interrupts also affect the flag in the way described. For example, some operator actions on the smartPAD are processed internally in the system by means of interrupts and can thus cause this abnormal response.

**Syntax**INTERRUPT *Action* <Number>**Explanation of the syntax**

Element	Description
<i>Action</i>	<ul style="list-style-type: none"> <li>■ DISABLE: Disables an active interrupt.</li> <li>■ ENABLE: Enables a disabled interrupt.</li> </ul>
<i>Number</i>	<p>Type: INT</p> <p>Number (= priority) of the interrupt to which the <i>Action</i> is to refer.</p> <p><i>Number</i> can be omitted. In this case, DISABLE or ENABLE refers to all active interrupts.</p>

If DISABLE *Number* or ENABLE *Number* is applied to an inactive interrupt, this is impermissible and the robot controller generates an error message.

**Example 1**

The precondition for the sequence shown in the illustration is that the interrupt has been declared and is active:

```
INTERRUPT DECL 1 WHEN $IN[7]==TRUE DO UP1()
...
INTERRUPT ON 1
```

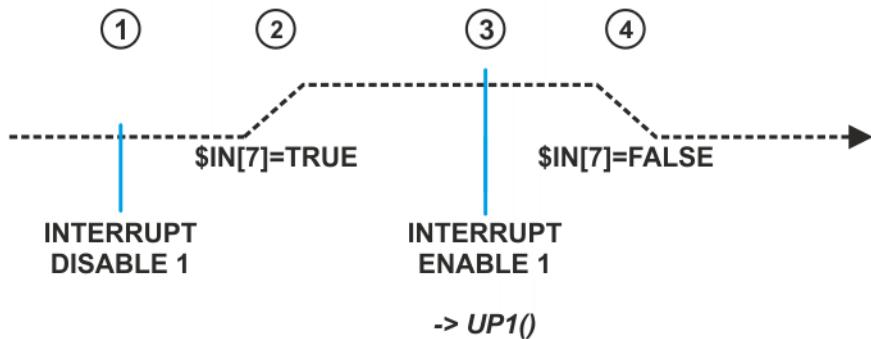


Fig. 11-12: Example 1

Item	Description
1	The interrupt is disabled.
2	The defined event occurs: input 7 is set to TRUE. The interrupt does not respond, as it is disabled.
3	The interrupt is re-enabled. It responds and calls the defined subprogram.
4	Input 7 is set back to FALSE.

**Example 2**

The precondition for the sequence shown in the illustration is that the interrupt has been declared and is active:

```
INTERRUPT DECL 1 WHEN $IN[7]==TRUE DO UP1()
...
INTERRUPT ON 1
```

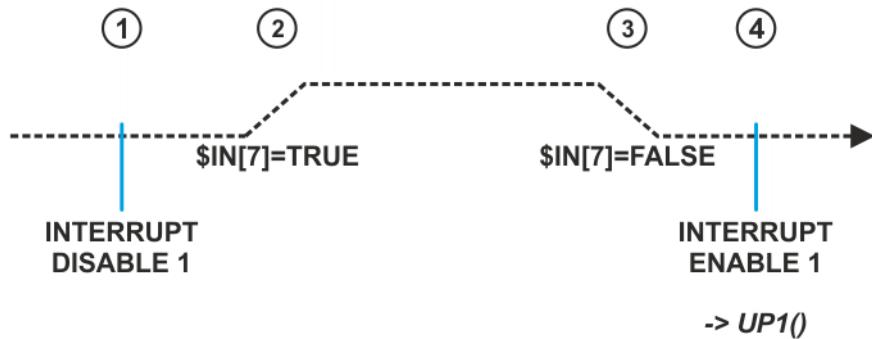


Fig. 11-13: Example 2

Item	Description
1	The interrupt is disabled.
2	The defined event occurs: input 7 is set to TRUE. The interrupt does not respond, as it is disabled.
3	Input 7 is set back to FALSE.
4	The interrupt is re-enabled. It responds and calls the defined subprogram. This is despite the fact that \$IN[7] is no longer TRUE. The decisive factor is that the defined event occurred (at least) once while the interrupt was disabled.

**Example 3**

The precondition for the sequence shown in the illustration is that the interrupts have been declared and are active:

```

INTERRUPT DECL 1 WHEN $IN[7]==TRUE DO UP1()
INTERRUPT DECL 2 WHEN $IN[8]==TRUE DO UP2()
...
INTERRUPT ON 1
INTERRUPT ON 2

```

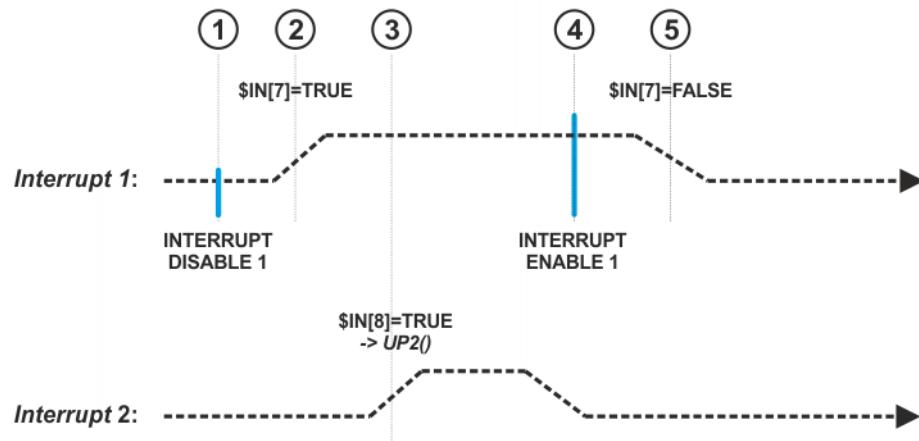


Fig. 11-14: Example 3

Item	Description
1	Interrupt 1 is disabled.
2	The defined event for interrupt 1 occurs: input 7 is set to TRUE. The controller now sets the event flag internally! Interrupt 1 does not respond to the event, as it is disabled.

Item	Description
3	The defined event for interrupt 2 occurs: input 8 is set to TRUE. Interrupt 2 responds and executes the defined action (= UP2). Due to execution of the action, the event flag is internally reset!
4	Interrupt 1 is re-enabled. It does not respond to the event retroactively, as the event flag is no longer set.
5	Input 7 is set back to FALSE.

**Example 4**

The precondition for the sequence shown in the illustration is that the interrupts have been declared and are active:

```
INTERRUPT DECL 1 WHEN $IN[7]==TRUE DO UP1()
INTERRUPT DECL 2 WHEN $IN[8]==TRUE DO UP2()
...
INTERRUPT ON 1
INTERRUPT ON 2
```

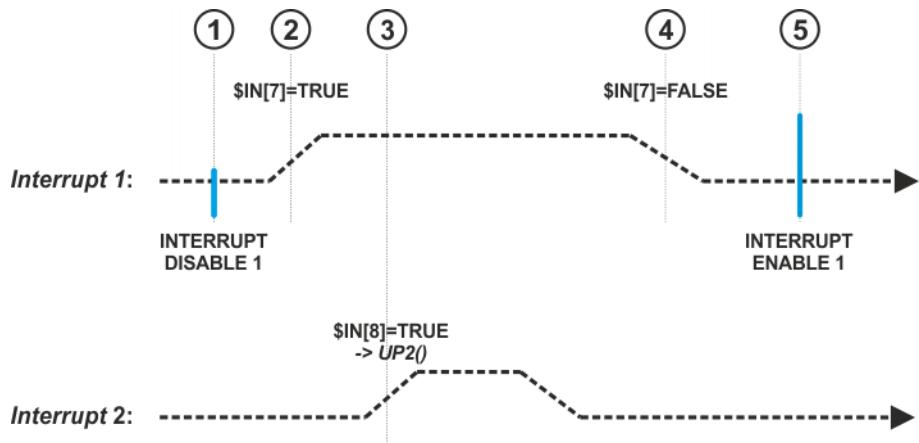


Fig. 11-15: Example 4

Item	Description
1	Interrupt 1 is disabled.
2	The defined event for interrupt 1 occurs: input 7 is set to TRUE. The controller now sets the event flag internally! Interrupt 1 does not respond to the event, as it is disabled.
3	The defined event for interrupt 2 occurs: input 8 is set to TRUE. Interrupt 2 responds and executes the defined action (= UP2). Due to execution of the action, the event flag is internally reset!
4	Input 7 is set back to FALSE.
5	Interrupt 1 is re-enabled. It does not respond to the event retroactively, as the event flag is no longer set.

#### 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: ([>>> 11.11.1 "INTERRUPT ... DECL ... WHEN ... DO ... : declaring an interrupt"](#) Page 425)

BRAKE in interrupt program: ([>>> 11.11.2 "INTERRUPT ON/OFF: activating or deactivating an interrupt"](#) Page 428)

### 11.11.5 RESUME: aborting 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.

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.)

■ **SCIRC, spline block:**

If the first motion instruction after RESUME is a SCIRC or a spline block, it will differ from how it was originally planned.

Reason: Following a RESUME statement, the robot is not situated at the original start point of the motion.

■ **CIRC:**

If the first motion instruction after RESUME is a CIRC motion, this is executed as LIN.

Reason: The robot controller internally changes the motion type to a constant.



**WARNING** Recommendation: After a RESUME, only use the motions SLIN, SPTP, LIN or PTP.

- If, however, 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 find itself when the RESUME statement is executed.
- If, however, an SCIRC or spline block is programmed as the first motion after RESUME, the robot must be able to safely carry out the motion, even with the modified route, from any position in which it could find itself 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
<i>Position</i>	Type: INT; variable or constant Reference point of the trigger <ul style="list-style-type: none"> <li>■ 0: Start point</li> <li>■ 1: End point</li> </ul> <p>(&gt;&gt;&gt; "Reference point for approximate positioning" Page 436)</p>
<i>Time</i>	Type: REAL; variable, constant or function; unit: ms Shift in time relative to <i>Position</i> . If no offset is desired, set <i>Time</i> = 0. <ul style="list-style-type: none"> <li>■ Negative value: Offset towards the start of the motion</li> <li>■ Positive value: Offset towards the end of the motion</li> </ul> <p>(&gt;&gt;&gt; "Max. offset" Page 437)</p> <p>If <i>Time</i> calls a function, the function is subject to constraints. (&gt;&gt;&gt; 11.12.3 "Constraints for functions in the trigger" Page 445)</p>
<i>Statement</i>	Possible: <ul style="list-style-type: none"> <li>■ Assignment of a value to a variable <b>Note:</b> There must be no runtime variable on the left-hand side of the assignment.</li> <li>■ OUT statement; PULSE statement; CYCFLAG statement</li> <li>■ Subprogram call. In this case, <i>Priority</i> must be specified.</li> </ul>
<i>Priority</i>	Type: INT; variable or constant Priority of the trigger. Only relevant if <i>Statement</i> 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.

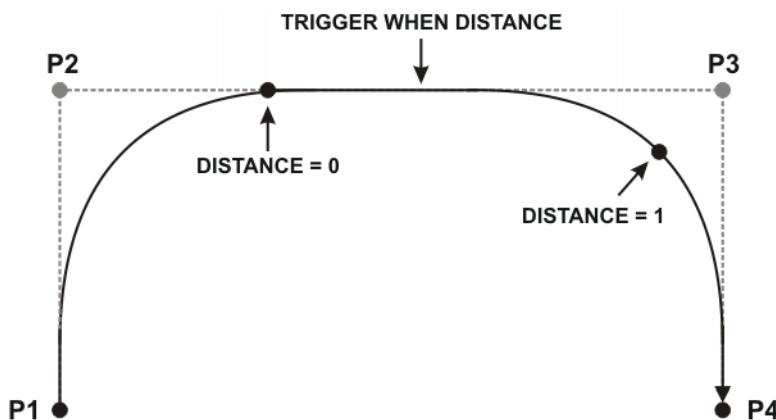


If a trigger calls a subprogram, it counts as an active interrupt until the subprogram has been executed. Up to 16 interrupts may be active at any one time.

## 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. 11-16: 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 off-set ...	Maximum positive off-set ...
DISTANCE = 0	--- (No negative offset possible.)	<ul style="list-style-type: none"> <li>■ Up to end point</li> <li>■ If the end point is approximated: up to the start of the approximate positioning arc</li> </ul>
DISTANCE = 1 and End point = exact position- ing point	<ul style="list-style-type: none"> <li>■ Up to start point</li> <li>■ If the start point is approximated: up to the end of the approximate positioning arc</li> </ul>	--- (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. 11-17 ):**

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

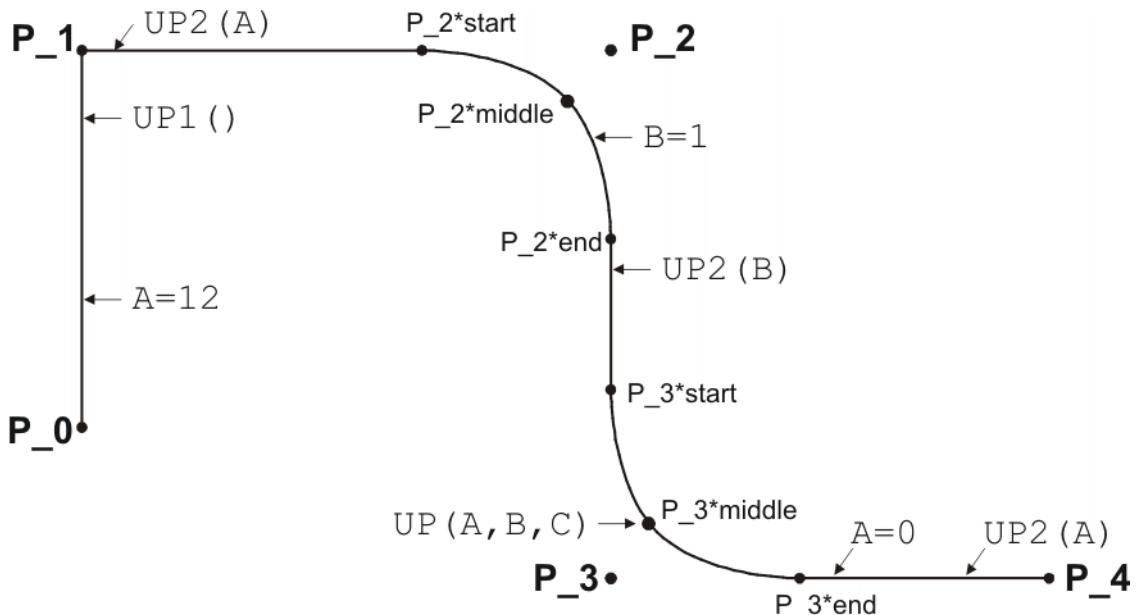


Fig. 11-17: Example of TRIGGER WHEN DISTANCE

### 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 "Constraints for functions in the trigger" Page 445)

## Explanation of the syntax

Element	Description
ONSTART	<p>Reference point of the trigger</p> <ul style="list-style-type: none"> <li>■ With ONSTART: Start point</li> <li>■ Without ONSTART: End point</li> </ul> <p>(&gt;&gt;&gt; 11.12.2.1 "Reference point for approximate positioning – overview" Page 442)</p>
<i>Distance</i>	<p>Type: REAL; variable, constant or function; unit: mm (except in the case of PTP splines without unit)</p> <p>Shift in space relative to the reference point. If no shift in space is desired, set <i>Distance</i> = 0.</p> <ul style="list-style-type: none"> <li>■ Negative value: Offset towards the start of the motion</li> <li>■ Positive value: Offset towards the end of the motion</li> </ul> <p>(&gt;&gt;&gt; "Max. offset" Page 440)</p>
<i>Time</i>	<p>Type: REAL; variable, constant or function; unit: ms</p> <p>Shift in time relative to <i>Distance</i>. If no shift in time is desired, set <i>Time</i> = 0.</p> <ul style="list-style-type: none"> <li>■ Negative value: Offset towards the start of the motion</li> <li>■ Positive value: Trigger is switched after <i>Time</i> has elapsed.</li> </ul> <p>(&gt;&gt;&gt; "Max. offset" Page 440)</p>

Element	Description
<i>Statement</i>	<p>Possible:</p> <ul style="list-style-type: none"> <li>■ Assignment of a value to a variable <b>Note:</b> There must be no runtime variable on the left-hand side of the assignment.</li> <li>■ OUT statement; PULSE statement; CYCFLAG statement</li> <li>■ Subprogram call. In this case, <i>Priority</i> must be specified.</li> </ul>
<i>Priority</i>	<p>Type: INT; variable or constant</p> <p>Priority of the trigger. Only relevant if <i>Statement</i> calls a subprogram, and then obligatory.</p> <p>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.</p> <p>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.</p>



If a trigger calls a subprogram, it counts as an active interrupt until the subprogram has been executed. Up to 16 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 trigger 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: <ul style="list-style-type: none"> <li>■ If the start point is an approximated PTP point: Up to the end of the approximate positioning arc</li> <li>■ If the start point is a different approximated point: Up to the start of the approximate positioning arc</li> </ul>	If the end point is an approximate positioning point: <ul style="list-style-type: none"> <li>■ In the case of homogenous approximate positioning: up to the next exact positioning point after the TRIGGER statement</li> <li>■ 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. (&gt;&gt;&gt; 11.12.2.3 "Reference point for mixed approximate positioning (spline)" Page 444)</li> <li>■ In the case of mixed approximate positioning (LIN/CIRC/PTP): up to the start of the approximate positioning arc</li> </ul>

#### 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.

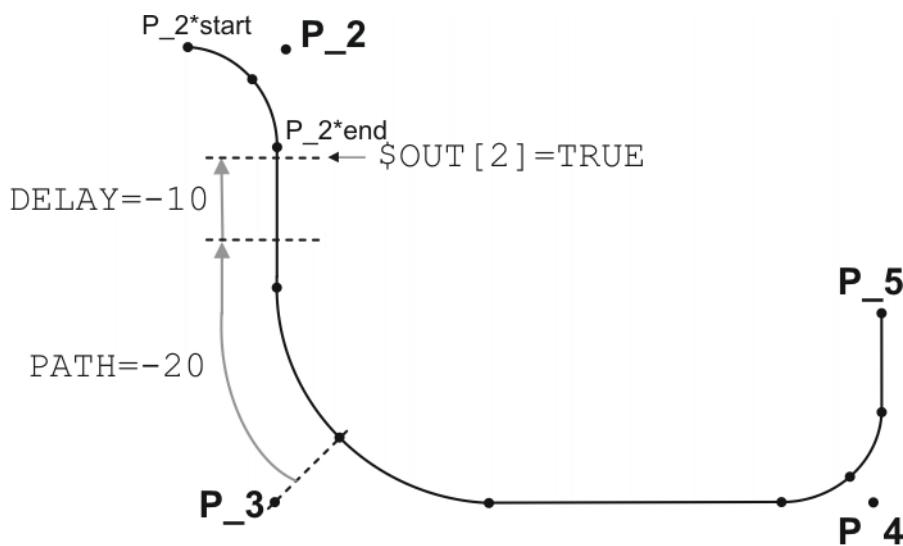


Fig. 11-18: Example of TRIGGER WHEN PATH

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.

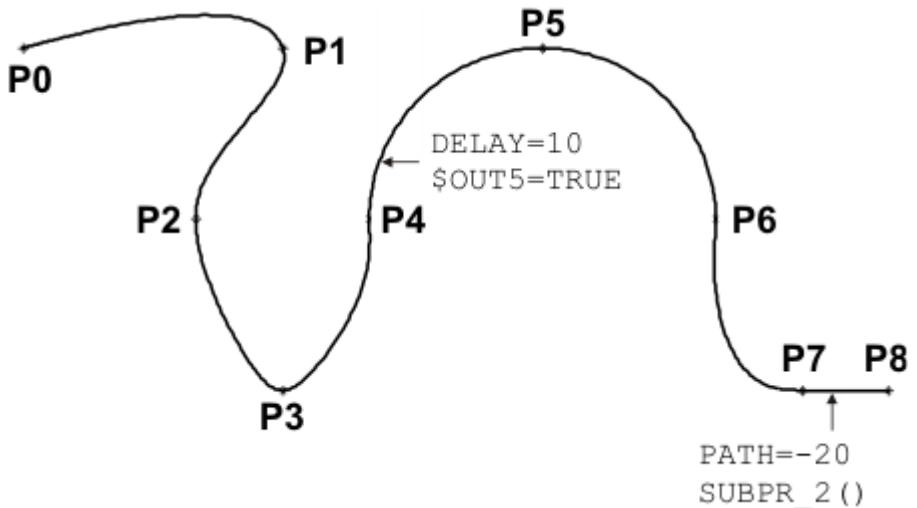


Fig. 11-19: Example of TRIGGER WHEN PATH (for spline)

#### 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 homogenous or mixed approximate positioning is being carried out.

##### Homogenous

##### Homogenous approximate positioning

- From a CP spline motion to a CP spline motion
- From a PTP spline motion to a PTP spline motion
- From a LIN or CIRC spline motion to a LIN or CIRC spline motion

Every spline motion can be a spline block or an individual instruction.

(>>> 11.12.2.2 "Reference point for homogenous approximate positioning"  
Page 443)

## Mixed

### Mixed approximate positioning

In this case, the position of the reference point also depends on whether the motions are spline motions or conventional motions.

- From a CP spline motion to a PTP spline motion or vice versa  
Every spline motion can be a spline block or an individual instruction.  
(>>> 11.12.2.3 "Reference point for mixed approximate positioning (spline)" Page 444)
- 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 445)

## 11.12.2 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
```

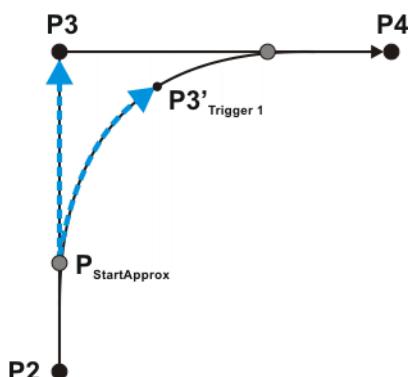
Triggers 1 and 2 both refer to P3. P3 is approximated. The robot controller transfers the point onto the approximate positioning arc by a distance corresponding to the approximation distance (= P3').

## 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  $P_{StartApprox} \rightarrow P3'_{Trigger 1}$ .



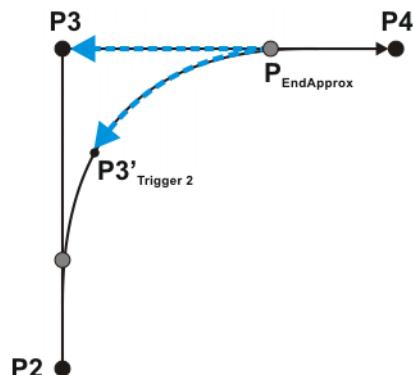
**Fig. 11-20: Trigger 1: Reference point for homogenous approximate positioning**

<b>P<sub>StartApprox</sub></b>	Start of the approximate positioning arc
<b>P3</b>	Reference point for exact positioning
<b>P3'<sub>Trigger 1</sub></b>	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. 11-21: Trigger 2: Reference point for homogenous approximate positioning**

<b>P<sub>EndApprox</sub></b>	End of the approximate positioning arc
<b>P3</b>	Reference point for exact positioning
<b>P3'<sub>Trigger 2</sub></b>	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.

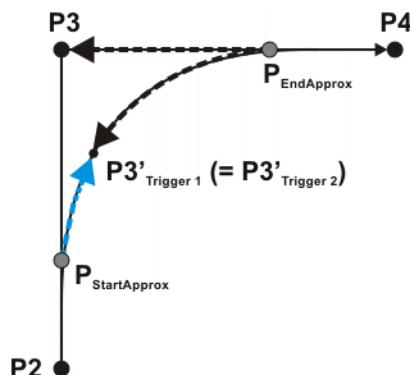
(>>> "Start point approximated" Page 444)

**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  $P_{StartApprox} \rightarrow P'_{Trigger 1}$  is generally shorter than  $P_{StartApprox} \rightarrow P_3$ .



**Fig. 11-22: Trigger 1: Reference point for mixed approximate positioning**

<b>P<sub>StartApprox</sub></b>	Start of the approximate positioning arc
<b>P<sub>EndApprox</sub></b>	End of the approximate positioning arc
<b>P<sub>3</sub></b>	Reference point for exact positioning
<b>P'<sub>3</sub></b>	Reference point for approximate positioning

If Trigger 1 were to be shifted to between  $P_{StartApprox}$  and  $P'_3$ , the exact position would be determined as follows:

The robot controller calculates the percentage of the distance  $P_{StartApprox} \rightarrow P_3$  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  $P_{StartApprox} \rightarrow P'_{Trigger 1}$

#### 11.12.2.4 Reference point for mixed approximate positioning (LIN/CIRC/PTP)

**Start point approximated**

**PTP-CP approximate positioning:**

The reference point is at the end of the approximate positioning arc.

**End point approximated**

**CP-PTP approximate positioning:**

The reference point is at the start of the approximate positioning arc.

#### 11.12.3 Constraints for functions in the trigger

The values for **DELAY** and **PATH** can be assigned using functions. The following constraints apply to these functions:

- The KRL program containing the function must have the attribute **Hidden**.  
(**>>> 7.4.2 "Displaying or modifying properties of files and folders"**  
Page 239)
- 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 Useful system variables for working with PATH triggers

##### 11.12.4.1 \$DIST\_NEXT

<b>Description</b>	\$DIST_NEXT specifies the length of the path from the current TCP position to the next taught point.  Type: REAL. Unit: <ul style="list-style-type: none"><li>■ For CP motions (spline and conventional): mm</li><li>■ For SPTP motions: No unit</li></ul> \$DIST_NEXT cannot be used for PTP motions. The value is always zero in this case.  \$DIST_NEXT is write-protected.
<b>Procedure</b>	\$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. <ol style="list-style-type: none"><li>1. Move to the position on the path where the switching point is to be located.</li><li>2. Read the system variable.</li><li>3. Program the trigger before the next point.<ul style="list-style-type: none"><li>■ Program the trigger without ONSTART.</li><li>■ Assign the value of the system variable to the PATH parameter.</li></ul></li></ol>

##### 11.12.4.2 \$DIST\_LAST

<b>Description</b>	\$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: <ul style="list-style-type: none"><li>■ For CP motions (spline and conventional): mm</li><li>■ For SPTP motions: No unit</li></ul> \$DIST_LAST cannot be used for PTP motions. The value is always zero in this case.  \$DIST_LAST is write-protected.
<b>Procedure</b>	\$DIST_LAST can be used as an aid for programming PATH triggers with ON-START. It can be used to determine the value that must be assigned to the PATH parameter. <ol style="list-style-type: none"><li>1. Move to the position on the path where the switching point is to be located.</li><li>2. Read the system variable.</li><li>3. Program the trigger after the previous point.<ul style="list-style-type: none"><li>■ Program the trigger with ONSTART.</li><li>■ Assign the value of the system variable to the PATH parameter.</li></ul></li></ol>

#### 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.

### 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
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, y 0, z 0, a 0, b 0, c 0}
		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.

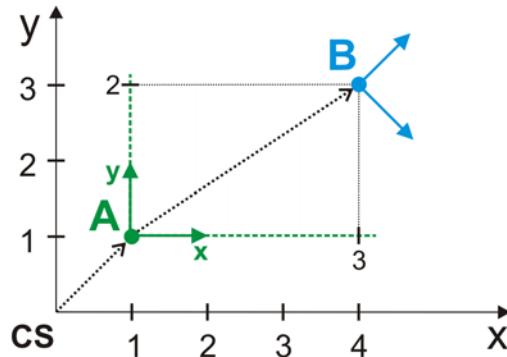
#### Sequence A:B

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. 11-23: R = A : B**

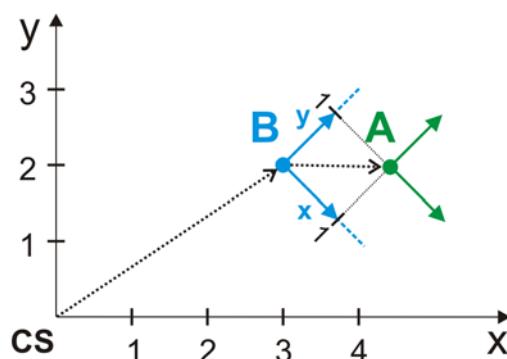
#### Sequence B:A

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. 11-24: 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.

Line	Description
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.

#### 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.

##### 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 [...]".

#### 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. 11-25: 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. 11-26: 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. 11-27: 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. 11-28: 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.

<b>Priority</b>	<b>Operator</b>
1	NOT; B_NOT
2	*; /
3	+; -
4	AND; B_AND
5	EXOR; B_EXOR
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

<b>Function</b>	<b>Range of values of argument</b>	<b>Range of values of result</b>
<b>ABS(X)</b> Absolute value	REAL_MIN...REAL_MAX	0 ... REAL_MAX
<b>SQRT(X)</b> Square root	0 ... REAL_MAX	0 ... REAL_MAX
<b>SIN(X)</b> Sine	REAL_MIN...REAL_MAX	-1 ... +1
<b>COS(X)</b> Cosine	REAL_MIN...REAL_MAX	-1 ... +1
<b>TAN(X)</b> Tangent	REAL_MIN...REAL_MAX	REAL_MIN...REAL_MAX
<b>ACOS(X)</b> Arc cosine	-1 ... +1	0 ... +180
<b>ATAN2(Y,X)</b> Arc tangent	REAL_MIN...REAL_MAX	-180 ... +180

Data type of all functions: REAL. Data type of all arguments: REAL.

### Absolute value

ABS(X) calculates the absolute value of X.

Example:

```
B = -3.4
A = 5*ABS (B) ;A=17.0
```

### Root

SQRT(X) calculates the square root of X.

Example:

```
A = SQRT(16.0801) ;A=4.01
```

### Sine

SIN(X) calculates the sine of angle X.

Example:

```
A = SIN(30) ;A=0,5
```

**Cosine**

COS(X) calculates the cosine of angle X.

Example:

B = 2*COS (45)	;B=1.41421356
----------------	---------------

**Tangent**

TAN(X) calculates the tangent of angle X.

Example:

C = TAN (45)	;C=1.0
--------------	--------

The tangent of the following absolute values is infinite:

- $\pm 90^\circ$
- $+90^\circ + k \cdot 180^\circ$  (where  $k = \pm$ 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  $SIN(X) = COS(90^\circ - X)$ .

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  $0^\circ$  and  $180^\circ$  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  $0^\circ$  and  $90^\circ$ . The arc tangent of negative values gives an angle between  $90^\circ$  and  $180^\circ$ .

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

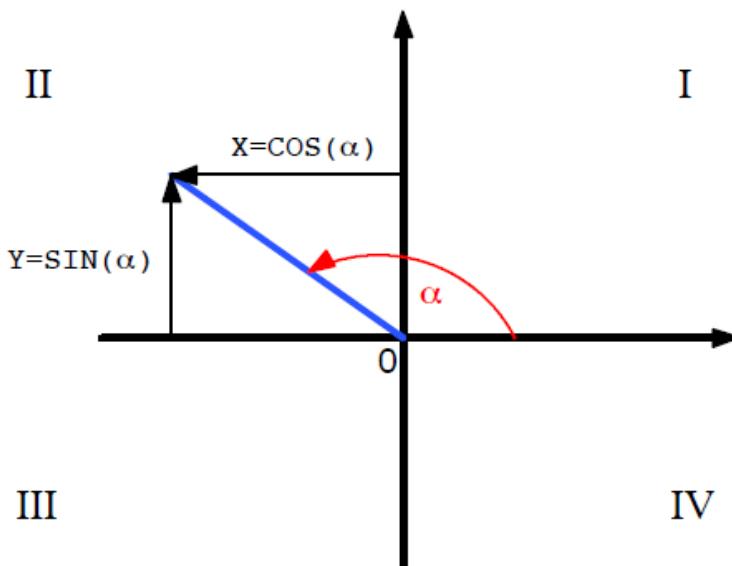


Fig. 11-29: Use of X and Y in the function ATAN(Y,X)

## 11.16 System functions

### 11.16.1 DELETE\_BACKWARD\_BUFFER()

#### 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. It can also be used in the trigger.

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.

#### Example 1

In the following example, it is still possible to start backward motion during the motion from P4 to P5. At P5, it is no longer possible:

```
PTP P4
PTP P5
DELETE_BACKWARD_BUFFER()
```

If it is desirable for backward motion already to be no longer possible during the motion from P4 to P5, this can be achieved by means of a trigger:

```
PTP P4
TRIGGER WHEN DISTANCE=0 DELAY=0 do DELETE_BACKWARD_BUFFER() PRIO=-1
PTP P5
```

#### Example 2

In the following example, the following actions are possible or not possible:

Action	Possible?
Move backwards to P6	Yes
Move backwards to P5	No
Move backwards from P5	Yes

```
PTP P3
PTP P4
```

```

PTP P5
TRIGGER WHEN DISTANCE=0 DELAY=0 do DELETE_BACKWARD_BUFFER() PRIO=-1
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, several triggers must be programmed:

```

PTP P3
TRIGGER WHEN DISTANCE=0 DELAY=0 do DELETE_BACKWARD_BUFFER() PRIO=-1
PTP P4
TRIGGER WHEN DISTANCE=0 DELAY=0 do DELETE_BACKWARD_BUFFER() PRIO=-1
PTP P5
TRIGGER WHEN DISTANCE=0 DELAY=0 do DELETE_BACKWARD_BUFFER() PRIO=-1
PTP P6
PTP P7

```

## 11.16.2 FORWARD()

### Description

The FORWARD function calculates the Cartesian position in space (BASE system) from the axis angles of the robot and the external axes.

The function can be used for diagnostic purposes.

- External axes are taken into account properly, both in the form of a Robot kinematic system and as a Base Kinematic system.
- The function can be called by means of the SHOWVAR or SETVAR command, or in a KRL program.

### Syntax

```
result = FORWARD (axis_values, err_status)
```

### Explanation of the syntax

Element	Description
<i>result</i>	Type: E6POS  Variable for the return value  Cartesian position relative to the BASE system.
<i>axis_values</i>	Type: E6AXIS  Transfer type: IN parameter  Axis angles of the robot for which the Cartesian position is to be calculated
<i>err_status</i>	Type: INT  Transfer type: OUT parameter  Setting whether the transferred axis angles are to be checked against software limit switches: <ul style="list-style-type: none"> <li>■ 0: All axis angles are tested. If they are not within the limits of the software limit switches, <i>err_status</i> returns an error code.</li> <li>■ &lt;&gt;0: Axis angles are not checked.</li> </ul>

### Error codes

The variable *err\_status* transfers the result of the calculation. If the calculation is not successful, it transfers a value that corresponds to an error code:

Value	Description
-4	The advance run variable \$TOOL is invalid.
-3	The advance run variable \$BASE is invalid.

<b>Value</b>	<b>Description</b>
-2	<i>err_status</i> does not yet have a valid value.
-1	Not all robot axis angles / required external axis angles have been defined.
0	Calculation was successful, no error.
1	Software limit switches violated with the specified axis angles.
2	Errors in the mathematical transformation

### 11.16.3 INV\_POS()

**Description** The function INV\_POS() inverts a position of type E6POS, POS or FRAME. The inversion is a frame inversion.  
The inversion only refers to the components X, Y, Z, A, B and C. Status and Turn, and the values for E1 to E6 (if available), remain constant.

**Syntax** *result* = INV\_POS (*position*)

**Explanation of the syntax**

<b>Element</b>	<b>Description</b>
<i>result</i>	Type: E6POS Variable for the return value Inverted position <ul style="list-style-type: none"><li>■ If the function was called with an argument of type E6POS, Status and Turn remain constant, as do E1 to E6.</li><li>■ If the function was called with an argument of type POS, E1 to E6 are not defined.</li><li>■ If the function was called with an argument of type FRAME, Status and Turn are not defined, nor are E1 to E6.</li></ul>
<i>position</i>	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.

### 11.16.4 INVERSE()

**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 Robot kinematic system and as a Base Kinematic system.
- The function can be called by means of the SHOWVAR or SETVAR command, or in a KRL program.

**Syntax** *result* = INVERSE (*position*, *start\_axis*, *err\_status*)

**Explanation of  
the syntax**

Element	Description
<i>result</i>	Type: E6AXIS  Variable for the return value  Axis angles at the transferred position
<i>position</i>	Type: E6POS  Transfer type: IN parameter  Cartesian position (with external axis angles if applicable) relative to the BASE system  The robot axis angles for this position are calculated.
<i>start_axis</i>	Type: E6AXIS  Transfer type: IN parameter  Axis angles of the robot at the start point of the motion
<i>err_status</i>	Type: INT  Transfer type: OUT parameter  Setting whether the transferred axis angles ( <i>start_axis</i> ) are to be checked against software limit switches: <ul style="list-style-type: none"><li>■ 0: All axis angles are tested. If they are not within the limits of the software limit switches, <i>err_status</i> returns an error code.</li><li>■ &lt;&gt;0: Axis angles are not checked.</li></ul> The calculated axis angles ( <i>result</i> ) are always checked against software limit switches.

**Using the start  
point**

The start point *start\_axis* is required in the following cases:

- The end point has no status value.

The system variable \$TARGET\_STATUS defines which status value the end point is to receive:

- \$TARGET\_STATUS=#SOURCE

The end point receives the same status as the start point. The status is calculated from the axis angles of the point *start\_axis*.

- \$TARGET\_STATUS=#BEST

The end point receives the status that enable the robot to cover the shortest possible distance from the start point to the end point in axis space.

- The end point has no turn value.

For each axis, the permissible turn value is calculated which results in the shortest path between the start point and end point. "Permissible" in this case means within the software limit switches.

- The end point is situated close to a singularity.

One axis angle must be specified and the one that is dependent on it is calculated. The system variable \$SINGUL\_POS [1...3] is used to set the angle that the end point is to receive:

- \$SINGUL\_POS [1...3]=0: The angle of the axis is set to 0 degrees.
- \$SINGUL\_POS [1...3]=1: The angle remains the same from the start point to the end point.

**Error codes**

The variable *err\_status* transfers the result of the calculation. If the calculation is not successful, it transfers a value that corresponds to an error code:

Value	Description
-4	The advance run variable \$TOOL is invalid.
-3	The advance run variable \$BASE is invalid.
-2	<i>err_status</i> does not yet have a valid value.
-1	Not all essential components of the <i>position</i> variable have been defined. S and T do not strictly need to be defined.
0	Calculation was successful, no error.
1	Software limit switches violated with the specified axis angles.
2	There are no axis angles with which the Cartesian end position can be reached.
3	There are axis angles with which the Cartesian end position can be reached, but the specified Turn would result in a software limit switch being violated.

**Example**

The KRL program KUE\_WEG calculates the status for axis 5 in such a way that the PTP motion of axes 4 and 5 between the start point and end point is as short as possible. It can be integrated into a motion program or used for the offline generation of Cartesian points.

The program is available as standard on every controller:

Directory	C:\KRC\UTIL\KUEWEG
File	kue_weg.src

### 11.16.5 ROB\_STOP() and ROB\_STOP\_RELEASE()

**Description**

ROB\_STOP() and ROB\_STOP\_RELEASE() can only be used in submit programs.

- ROB\_STOP() stops the robot and prevents further motions. This affects all possible motions, irrespective of whether they arise from program execution, manual jogging or command mode.
- ROB\_STOP\_RELEASE() cancels a blockade caused by ROB\_STOP().

If ROB\_STOP() is called several times in succession without a ROB\_STOP\_RELEASE() in between, only the first call has any effect. The subsequent calls have no effect, i.e. they do not trigger a stop or cause a message to be generated.

If ROB\_STOP\_RELEASE() is called without ROB\_STOP() being called first, this has no effect.

**Messages**

ROB\_STOP() triggers the following status message: *Robot stopped by submit*  
ROB\_STOP\_RELEASE() in a Test mode triggers the following acknowledgement message: *Ackn. Robot stopped by submit*

ROB\_STOP\_RELEASE() in an Automatic mode triggers no message.

**Syntax**

*result* = ROB\_STOP (stop\_type)

Explanation of the syntax	Element	Description
	<i>result</i>	<p>Type: BOOL Variable for the return value. Return value:</p> <ul style="list-style-type: none"> <li>■ TRUE: The stop has been executed.</li> <li>■ FALSE: An invalid parameter has been transferred for <i>stop_type</i>.</li> </ul>
	<i>stop_type</i>	<p>Type: ROB_STOP_T Transfer type: IN parameter Stop type to be used to stop the robot:</p> <ul style="list-style-type: none"> <li>■ #RAMP_DOWN: ramp stop</li> <li>■ #PATH_MAINTAINING: path-maintaining EMERGENCY STOP</li> </ul> <p>Other stop types are not possible.</p>

## ProConOS

The “Robot stop” function can also be used from ProConOS. The following functions are available for this:

- PLC\_ROB\_STOP()  
The desired stop type is defined with PLC\_ROB\_STOP\_RAMP\_DOWN or PLC\_ROB\_STOP\_PATH\_MAINT.
- PLC\_ROB\_STOP\_RELEASE()

The effect is the same, irrespective of whether a stop is triggered by submit or by ProConOS. ProConOS generates its own message texts; these are:

- *Robot stopped by SoftPLC ({Name of the task calling it})*
- *Ackn. Robot stopped by SoftPLC*

If a stop is requested by both submit and ProConOS, this is indicated in each case by a status message. A maximum of 2 status messages can thus be displayed for an executed stop. In this case, robot motion cannot be resumed until the blockade has been canceled by both submit and ProConOS.

If different stop types are requested by submit and ProConOS, the type actually carried out is generally the one that was requested first.

A stop triggered by ProConOS cannot be canceled by a submit program and vice versa.

### 11.16.6 SET\_BRAKE\_DELAY()

#### Description

The function SET\_BRAKE\_DELAY can be used to reduce the brake delay with reference to an individual point.

SET\_BRAKE\_DELAY is intended for use at the end of a cycle. When the robot stops there before the next cycle begins, SET\_BRAKE\_DELAY can be used to make the brakes close earlier, thereby also causing the drives to be deactivated sooner. Energy can be saved in this way.

#### Brake delay:

The brake delay is the time after which the brakes are applied when the robot (or the external axis) has reached an exact positioning point. It is irrelevant whether the exact positioning point was programmed as such, or whether it just works out that way because approximate positioning cannot be carried out.

If the robot stops at the point until the time has elapsed, e.g. at the end of the program, the brakes are applied. If the robot resumes motion before the time has elapsed, the brakes are not applied.

The generally applicable brake delay is defined in system variables. SET\_BRAKE\_DELAY can be used to set a lower value for an individual point, i.e. the brakes are applied earlier.

System variables for the generally applicable brake delay:

- \$BRK\_DEL\_COM:  
Brake delay for robot axes in command mode (= jogging) (default: 10,000 ms)
- \$BRK\_DEL\_PRO:  
Brake delay for robot axes in program mode (default: 20,000 ms)
- \$BRK\_DEL\_EX:  
Brake delay for external axes (default: 200 ms)  
\$BRK\_DEL\_EX only applies if the external axis mode is set (\$BRK\_MODE, bit 3 =1) and the external axis is not mathematically coupled. Otherwise, the brakes of the external axis respond in the same way as the robot axes and the corresponding delay times apply.



Further information about \$BRK\_MODE can be found in the documentation **Configuration of Kinematic Systems**.

## 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
<i>result</i>	<p>Type: INT</p> <p>Variable for the return value. The bits indicate the axes for which <i>delay</i> has been set.</p> <ul style="list-style-type: none"> <li>■ Bit n = 0: value was not set for this axis.</li> <li>■ Bit n = 1: value was set for this axis.</li> </ul> <p>The return value does not indicate whether the brakes were actually applied.</p>
<i>axes_nr</i>	<p>Type: INT</p> <p>Bit array for the axes for which <i>delay</i> is to be set.</p> <ul style="list-style-type: none"> <li>■ Bit n = 0: value is not set for this axis.</li> <li>■ Bit n = 1: value is set for this axis.</li> </ul> <p>The value can be specified in the program as an integer or using bit notation, e.g. "63" or "B111111" for "all robot axes".</p> <p>By default, the brakes of robot axes are applied simultaneously. In this case, the value applies to all robot axes, including those that are not specified here.</p> <p>If the brakes of external axes are applied simultaneously (dependent on \$BRK_MODE), the value applies to all external axes, including those that are not specified here. The brakes of master/slave axes are always applied simultaneously.</p>
<i>delay</i>	<p>Type: INT, unit: ms</p> <p>Desired delay. Range of values:</p> <ul style="list-style-type: none"> <li>■ <b>0</b> ... defined general brake delay</li> </ul> <p>If a higher value is defined, it is limited internally to the value of the relevant system variable: \$BRK_DEL_COM, \$BRK_DEL_PRO or \$BRK_DEL_EX</p> <p>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.</p>

Bit n	11 ...	5	4	3	2	1	0
Axis	E6 ...	A6	A5	A4	A3	A2	A1

**\$BRAKE\_SIG**

The state of the brakes (open or closed) can be displayed by means of the system variable \$BRAKE\_SIG.

(>>> "\$BRAKE\_SIG" Page 220)

**Example**

At the end of the following program, the brakes of the robot axes are to be applied as quickly as possible. For this reason, SET\_BRAKE\_DELAY(63, 0) has been programmed after the last point.

```

1 DEF my_test()
2 DECL INT my_result
3 DECL INT brake_state
...
4 PTP HOME Vel= 100 % DEFAULT
5 PTP P1 ...
...
```

```

6 PTP HOME Vel= 100 % DEFAULT
7 my_result = SET_BRAKE_DELAY(63, 0)
8 brake_state = $BRAKE_SIG
9 END

```

Line	Description
6	Last point in program
7	Here, the brake delay is set to 0 ms for all robot axes for the point in line 6.
8	The monitoring reveals that the brakes are (still) open at this point. The reason for this is that the brakes cannot close in 0 ms; they require a certain time to close for mechanical reasons. Once this time has elapsed, the brakes are closed.

### Negative example

It is not generally helpful to use SET\_BRAKE\_DELAY during a cycle. There are often no points at which the brakes are applied and where this operation would need to be accelerated. On the contrary, the advance run stop triggered by SET\_BRAKE\_DELAY would actually have a negative effect on the cycle time.

```

1 DEF my_test()
2 DECL INT my_result
...
3 PTP HOME Vel= 100 % DEFAULT
4 PTP P1 C_DIS ...
5 my_result = SET_BRAKE_DELAY(63, 0)
6 ;WAIT SEC 0.5
7 PTP P2 ...
...

```

Line	Description
5	Here, the brake delay is set to 0 ms for all robot axes for P1. P1 is programmed with approximate positioning. Since SET_BRAKE_DELAY triggers an advance run stop, the motion to P1 is carried out with exact positioning.  The brakes do not close at P1, however. The reason for this is that the brakes would need the time mechanically required to close. However, on reaching P1, the robot controller immediately starts the next motion. The brakes are thus not applied, even though the delay is set to 0 ms.
6	By contrast, if this line were uncommented, the brakes would close.  The robot controller would not immediately start the next motion on reaching P1, but stop at P1 for 0.5 s. This would allow time for the brakes to be applied.

## 11.16.7 VARSTATE()

### Description

VARSTATE() can be used to monitor the state of a variable.

VARSTATE() is a function with a return value of type VAR\_STATE. VAR\_STATE is an enumeration type that is defined as follows in the system:

```
ENUM VAR_STATE DECLARED, INITIALIZED, UNKNOWN
```

VARSTATE is defined as follows in the system:

```
VAR_STATE VARSTATE (CHAR VAR_STR[80] :IN)
```

**Example 1**

```

DEF PROG1()
INT MYVAR
...
IF VARSTATE ("MYVAR") ==#UNKNOWN THEN
    $OUT[11]=TRUE
ENDIF
...
IF VARSTATE ("MYVAR") ==#DECLARED THEN
    $OUT[12]=TRUE
ENDIF
...
IF VARSTATE ("ANYVAR") ==#UNKNOWN THEN
    $OUT[13]=TRUE
ENDIF
...
MYVAR=9
...
IF VARSTATE ("MYVAR") ==#DECLARED THEN
    $OUT[14]=TRUE
ENDIF
...
IF VARSTATE ("MYVAR") ==#INITIALIZED THEN
    $OUT[15]=TRUE
ENDIF
...
END

```

**Explanation of the state monitoring:**

- The first IF condition is false, as MYVAR has already been declared. Output 11 is not set.
- The second IF condition is true, as MYVAR has been declared. Output 12 is set.
- The third IF condition is true, on the condition that there is also no variable with the name ANYVAR in \$CONFIG.DAT. Output 13 is set.
- The fourth IF condition is false, as MYVAR has not only been declared, but has also already been initialized here. Output 14 is not set.
- The fifth IF condition is true, as MYVAR has been initialized. Output 15 is set.

**Example 2**

```

DEF PROG2()
INT MYVAR
INT YOURVAR
DECL VAR_STATE STATUS
...
STATUS=VARSTATE ("MYVAR")
UP()
...
STATUS=VARSTATE ("YOURVAR")
UP()
...
END

```

```

DEF UP()
...
IF VARSTATE ("STATUS") ==#DECLARED THEN
    $OUT[100]=TRUE
ENDIF
...
END

```

Explanation of the state monitoring:

In this example, the state is monitored indirectly, i.e. via an additional variable. The additional variable must be of type VAR\_STATE. The keyword DECL must not be omitted in the declaration. The name of the additional variable may be freely selected. In this example it is STATUS.

## 11.17 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 correction function.

The functions can be used within IF branches without the return value being explicitly assigned to a variable.

### 11.17.1 Converting a string variable to a different data type

<b>Description</b>	The functions of type StrTo[...] can be used to convert string variables to a different data type. The following functions are declared in KRL:								
	<ul style="list-style-type: none"> <li>■ BOOL StrToAXIS (CHAR strValue[256], AXIS value)</li> <li>■ BOOL StrToBOOL (CHAR strValue[256], BOOL value)</li> <li>■ BOOL StrToE3AXIS (CHAR strValue[256], E3AXIS value)</li> <li>■ BOOL StrToE6AXIS (CHAR strValue[256], E6AXIS value)</li> <li>■ BOOL StrToE3POS (CHAR strValue[256], E3POS value)</li> <li>■ BOOL StrToE6POS (CHAR strValue[256], E6POS value)</li> <li>■ BOOL StrToFRAME (CHAR strValue[256], FRAME value)</li> <li>■ BOOL StrToInt (CHAR strValue[256], INT value)</li> <li>■ BOOL StrToPOS (CHAR strValue[256], POS value)</li> <li>■ BOOL StrToREAL (CHAR strValue[256], REAL value)</li> <li>■ BOOL StrToString (CHAR strValue[256], STRING value)</li> </ul>								
<b>Syntax</b>	StrToAXIS is shown here as an example for the type StrTo[...]:								
	<code>success = StrToAXIS (string, value)</code>								
<b>Explanation of the syntax</b>	StrToAXIS is explained here as an example for the type StrTo[...]: The other functions are treated analogously.								
	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr style="background-color: #cccccc;"> <th style="text-align: left; padding: 2px;">Element</th> <th style="text-align: left; padding: 2px;">Description</th> </tr> </thead> <tbody> <tr> <td style="padding: 2px;">success</td> <td style="padding: 2px;">           Type: BOOL            Variable for the return value  <ul style="list-style-type: none"> <li>■ TRUE: Conversion successful</li> <li>■ FALSE: Conversion not successful</li> </ul> </td></tr> <tr> <td style="padding: 2px;">string</td> <td style="padding: 2px;">           Type: CHAR            String variable that is to be converted to a different data type            If <i>string</i> represents an aggregate, the individual components must be separated by commas.         </td></tr> <tr> <td style="padding: 2px;">value</td> <td style="padding: 2px;">           Type: AXIS            Variable for the value after conversion         </td></tr> </tbody> </table>	Element	Description	success	Type: BOOL Variable for the return value <ul style="list-style-type: none"> <li>■ TRUE: Conversion successful</li> <li>■ FALSE: Conversion not successful</li> </ul>	string	Type: CHAR String variable that is to be converted to a different data type If <i>string</i> represents an aggregate, the individual components must be separated by commas.	value	Type: AXIS Variable for the value after conversion
Element	Description								
success	Type: BOOL Variable for the return value <ul style="list-style-type: none"> <li>■ TRUE: Conversion successful</li> <li>■ FALSE: Conversion not successful</li> </ul>								
string	Type: CHAR String variable that is to be converted to a different data type If <i>string</i> represents an aggregate, the individual components must be separated by commas.								
value	Type: AXIS Variable for the value after conversion								

#### Example 1

```

1 DECL BOOL success, value
2 ...
3 success=StrToBOOL("1", value)
4 success=StrToBOOL("TRUE", value)

```

Line	Description
3	Conversion is not possible, as the string variable "1" does not match the data type BOOL. The return value <i>success</i> is FALSE.
4	Conversion is possible. The return value <i>success</i> is TRUE. The value after conversion is TRUE.

**Example 2**

```

1 DECL FRAME value
2 DECL BOOL success
3 ...
4 success=StrToFRAME("{X 10, Y 50, C 45}",value)

```

Line	Description
4	Conversion is possible. The return value <i>success</i> is TRUE. The value after conversion is "X 10, Y 50, C 45". Missing frame components are not initialized and thus remain without a value. If there are already components present in the target variable, they are deleted.

**Example 3**

```

1 DECL AXIS value
2 DECL BOOL success
3 ...
4 success=StrToAXIS("{A1 45}",value)
5 success=StrToAXIS("{E1 100}",value)
6 success=StrToAXIS("{A1 90, E1 100}",value)

```

Line	Description
4	Conversion is possible, as A1 is a component of AXIS. The return value <i>success</i> is TRUE.
5	Conversion is not possible, as E1 is not a component of AXIS. The return value <i>success</i> is FALSE.
6	Conversion is possible. The value of E1 is lost. The return value <i>success</i> is TRUE.

**11.17.2 String variable length in the declaration****Description**

The function `StrDeclLen()` determines the length of a string variable according to its declaration in the declaration section of a program.

**Syntax**

`Length = StrDeclLen (StrVar[])`

**Explanation of the syntax**

Element	Description
Length	Type: INT  Variable for the return value. Return value: Length of the string variable as declared in the declaration section
StrVar[]	Type: CHAR array  String variable whose length is to be determined  Since the string variable <code>StrVar[ ]</code> 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 String variable length after initialization

**Description** The function `StrLen()` determines the length of the character string of a string variable as defined in the initialization section of the program.

**Syntax** `Length = StrLen(StrVar)`

**Explanation of the syntax**

Element	Description
Length	Type: INT  Variable for the return value. Return value: Number of characters currently assigned to the string variable
StrVar	Type: CHAR  Character string or variable whose length is to be determined

**Example**

```
1 CHAR PartA[50]
2 INT AB
...
3 PartA[] = "This is an example"
4 AB = StrLen(PartA[])
```

Line	Description
4	AB = 18

### 11.17.4 Deleting the contents of a string variable

**Description** The function `StrClear()` deletes the contents of a string variable.

**Syntax** `Result = StrClear(StrVar[])`

**Explanation of the syntax**

Element	Description
Result	Type: BOOL  Variable for the return value. Return value: <ul style="list-style-type: none"><li>■ The contents of the string variable have been deleted: TRUE</li><li>■ The contents of the string variable have not been deleted: FALSE</li></ul>
StrVar[]	Type: CHAR array  Variable whose character string is to be deleted

**Example**

```
IF (NOT StrClear($Loop_Msg[])) THEN
HALT
ENDIF
```

The function can be used within IF branches without the return value being explicitly assigned to a variable. This applies to all functions for editing string variables.

### 11.17.5 Extending a string variable

**Description** The function `StrAdd()` can be used to expand a string variable with the contents of another string variable.

**Syntax** `Sum = StrAdd( StrDest[], StrToAdd[])`

**Explanation of the syntax**

Element	Description
Sum	Type: INT  Variable for the return value. Return value: Sum of <code>StrDest[ ]</code> and <code>StrToAdd[ ]</code>  If the sum is longer than the previously defined length of <code>StrDest[ ]</code> , 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 <code>StrDest[ ]</code> 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 Searching a string variable

**Description** The function `StrFind()` can be used to search a string variable for a character string.

**Syntax** `Result = StrFind( StartAt, StrVar[], StrFind[], CaseSens)`

**Explanation of the syntax**

Element	Description
Result	Type: INT  Variable for the return value. Return value: Position of the first character found. If no character is found, the return value is 0.
StartAt	Type: INT  The search is started from this position.
StrVar[]	Type: CHAR array  The string variable to be searched

Element	Description
StrFind[]	Type: CHAR array The character string that is being looked for.
CaseSens	<ul style="list-style-type: none"> <li>■ #CASE_SENS: Upper and lower case are taken into consideration.</li> <li>■ #NOT_CASE_SENS: Upper and lower case are ignored.</li> </ul>

**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 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: <ul style="list-style-type: none"><li>■ The character strings match: TRUE</li><li>■ The character strings do not match: FALSE</li></ul>
StrComp1[]	Type: CHAR array String variable that is compared with StrComp2[].
StrComp2[]	Type: CHAR array String variable that is compared with StrComp1[].
CaseSens	<ul style="list-style-type: none"> <li>■ #CASE_SENS: Upper and lower case are taken into consideration.</li> <li>■ #NOT_CASE_SENS: Upper and lower case are ignored.</li> </ul>

**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

Line	Description
6	B = FALSE
7	B = FALSE

### 11.17.8 Copying a string variable

**Description** The function `StrCopy()` can be used to copy the contents of a string variable to another string variable.

**Syntax** `Copy = StrCopy (StrDest[], StrSource[])`

**Explanation of the syntax**

Element	Description
Copy	Type: BOOL Variable for the return value. Return value: <ul style="list-style-type: none"><li>■ The string variable was copied successfully: TRUE</li><li>■ The string variable was not copied: FALSE</li></ul>
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 interpreter

- Function**
- 2 tasks run in parallel on the robot controller:
- Robot interpreter  
The motion program runs in the robot interpreter.
  - Submit interpreter  
A SUB program runs in the submit interpreter.  
A SUB program 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.
- The submit interpreter starts automatically when the robot controller is switched on. The program SPS.SUB is started.
- The submit interpreter can be stopped or deselected manually and can also be restarted.
- SUB programs are always files with the extension \*.SUB. The program SPS.SUB can be edited and further SUB programs can be created.



Submit interpreters must not be used for time-critical applications! A PLC must be used in such cases. Reasons:

- The submit interpreters share system resources with the robot interpreter, which has the higher priority. Submit interpreters are thus not executed at the robot controller's interpolation cycle rate of 12 ms. Furthermore, the runtime of the submit interpreters is irregular.
- The runtime of the submit interpreters is influenced by the number of lines in the SUB program. Even comment lines and blank lines have an effect.



If a system file, e.g. \$config.dat or \$custom.dat, is modified in such a way that errors are introduced, the Submit interpreter is automatically deselected. Once the error in the system file has been rectified, the Submit interpreter must be reselected manually.

- Display**
- The program SPS.SUB is located in the directory R1\System. This directory is visible in the user group Expert or higher.

In the Navigator, SUB programs are indicated by the following symbol:



By default, the execution of a selected SUB program is not displayed. This can be changed using the system variable \$INTERPRETER. The SUB program can only be displayed, however, if a motion program is selected at the same time.

\$INTERPRETER	Description
1	The selected motion program is displayed in the editor (default).
0	The selected SUB program is displayed in the editor.

## 12.2 Manually stopping or deselecting the Submit interpreter

<b>Precondition</b>	<ul style="list-style-type: none"> <li>■ User group "Expert".</li> <li>■ Operating mode T1 or T2.</li> </ul>						
<b>Procedure</b>	<ul style="list-style-type: none"> <li>■ In the main menu, select <b>Configuration &gt; SUBMIT interpreter &gt; Stop or Deselect.</b></li> </ul>						
<b>Alternative procedure</b>	<ul style="list-style-type: none"> <li>■ In the status bar, touch the <b>Submit interpreter</b> status indicator. A window opens. Select <b>Stop</b> or <b>Deselect</b>.</li> </ul>						
<b>Description</b>	<table border="1"> <thead> <tr> <th>Command</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td><b>Stop</b></td> <td>The submit interpreter is stopped. When it is restarted, the SUB program is resumed at the point at which it was stopped.</td> </tr> <tr> <td><b>Deselect</b></td> <td>The submit interpreter is deselected.</td> </tr> </tbody> </table>	Command	Description	<b>Stop</b>	The submit interpreter is stopped. When it is restarted, the SUB program is resumed at the point at which it was stopped.	<b>Deselect</b>	The submit interpreter is deselected.
Command	Description						
<b>Stop</b>	The submit interpreter is stopped. When it is restarted, the SUB program is resumed at the point at which it was stopped.						
<b>Deselect</b>	The submit interpreter is deselected.						

Once the submit interpreter has been stopped or deselected, the corresponding icon in the status bar is red or gray.

Icon	Color	Description
	Red	The submit interpreter has been stopped.
	Gray	The submit interpreter is deselected.

## 12.3 Manually starting the Submit interpreter

<b>Precondition</b>	<ul style="list-style-type: none"> <li>■ User group "Expert"</li> <li>■ Operating mode T1 or T2</li> <li>■ Submit interpreter has been stopped or deselected.</li> </ul>
<b>Procedure</b>	<ul style="list-style-type: none"> <li>■ In the main menu, select <b>Configuration &gt; SUBMIT interpreter &gt; Select/Start.</b></li> </ul>
<b>Alternative procedure</b>	<ul style="list-style-type: none"> <li>■ In the status bar, touch the <b>Submit interpreter</b> status indicator. A window opens. Select <b>Select/Start</b>.</li> </ul>
<b>Description</b>	<p>If the submit interpreter is deselected, the command <b>Start/Select</b> selects the program SPS.SUB.</p> <p>If the submit interpreter has been stopped, the command <b>Start/Select</b> resumes the selected program at the point at which it was stopped.</p> <p>Once the submit interpreter has been started, the corresponding icon in the status bar is green.</p>

Icon	Color	Description
	Yellow	The submit interpreter is selected. The block pointer is situated on the first line of the selected SUB program.
	Green	A SUB program is selected and running.

## 12.4 Editing the program SPS.SUB

**Description** The following folds are available for user-defined adaptations in the program SPS.SUB:

- 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.

**Precondition**

- The program sps.sub is not selected or has been stopped.
- User group “Expert”

**Procedure**

1. Select the program SPS.SUB in the Navigator 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**.
4. The program SPS.SUB can now be started via the main menu with **Configuration > SUBMIT interpreter > Select/Start**.

**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.

## 12.5 Creating a new SUB program

### Precondition

- “Expert” user group

### 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
2INI
3
4LOOP
5USER PLC
6ENDLOOP
7USER 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.

## 12.6 Programming

### KRL code

Almost all KRL instructions can be used in a SUB program. The following statements are not possible, however:

- 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.

Motion commands for external axes, on the other hand, can be used in a SUB program. Example:

```
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.

WAIT statements or wait loops have not been used here as they stop the cycle.

### System variables

The submit interpreter has 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 program is started and the submit interpreter stops.

System variables that are frequently required in SUB programs:

\$MODE_OP = Value	
Value	Description
#T1	Robot controller is in T1 mode.
#T2	Robot controller is in T2 mode.
#AUT	Robot controller is in Automatic mode.
#EX	Robot controller is in Automatic External mode.
#INVALID	Robot controller has no defined state.

\$OV_PRO = Value		
Element	Data type	Description
Value (%)	INT	Program override value

Example:

If the programmed velocity is not reached, output 2 is set to FALSE.

```
...
IF (( $MODE_OP == #T1) OR ($OV_PRO < 100)) THEN
$OUT[2] = FALSE
ENDIF
...
```

**WARNING**

In the test modes, \$OV\_PRO must not be written to by the 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 the 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.

**Inputs/outputs**

The submit interpreter can access the inputs and outputs of the robot controller.

**WARNING**

No check is made to see if 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.

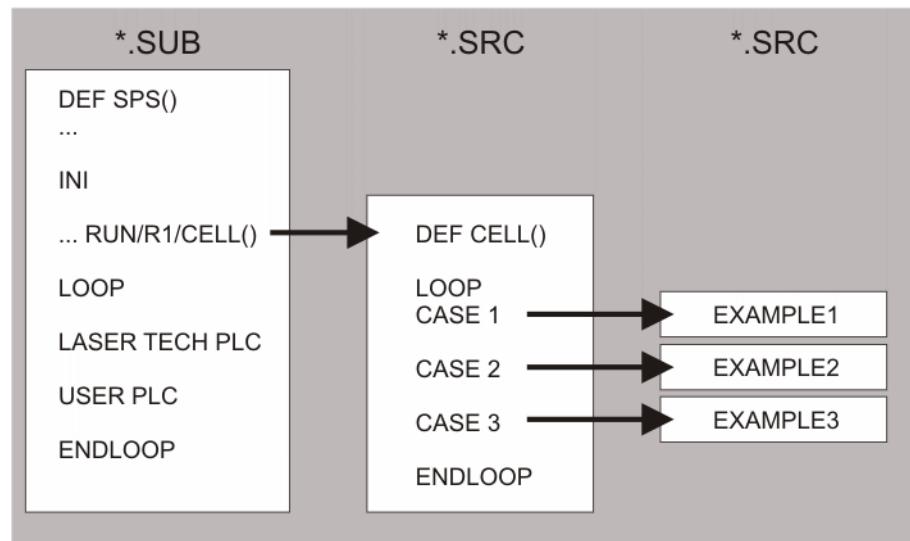
**Subprograms**

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 called from the program SPS.SUB with a CWRITE statement and RUN. The call only takes effect in the case of a cold start.

**KR C**

**Fig. 12-1: sps.sub selects CELL.SRC in the robot interpreter**



Further information about the program CELL.SRC can be found in this documentation.

(>>> 6.19.1 "Configuring CELL.SRC" Page 196)

Further information about CWRITE statements can be found in the Expert documentation CREAD/CWRITE.

**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.



## 13 Diagnosis

### 13.1 Logbook

#### 13.1.1 Displaying the logbook

The operator actions on the smartPAD are automatically logged.

##### Procedure

- In the main menu, select **Diagnosis > Logbook > Display**.

The following tabs are available:

- Log ([>>> 13.1.2 "Log" tab Page 481](#))
- Filter ([>>> 13.1.3 "Filter tab" Page 482](#))

#### 13.1.2 "Log" tab

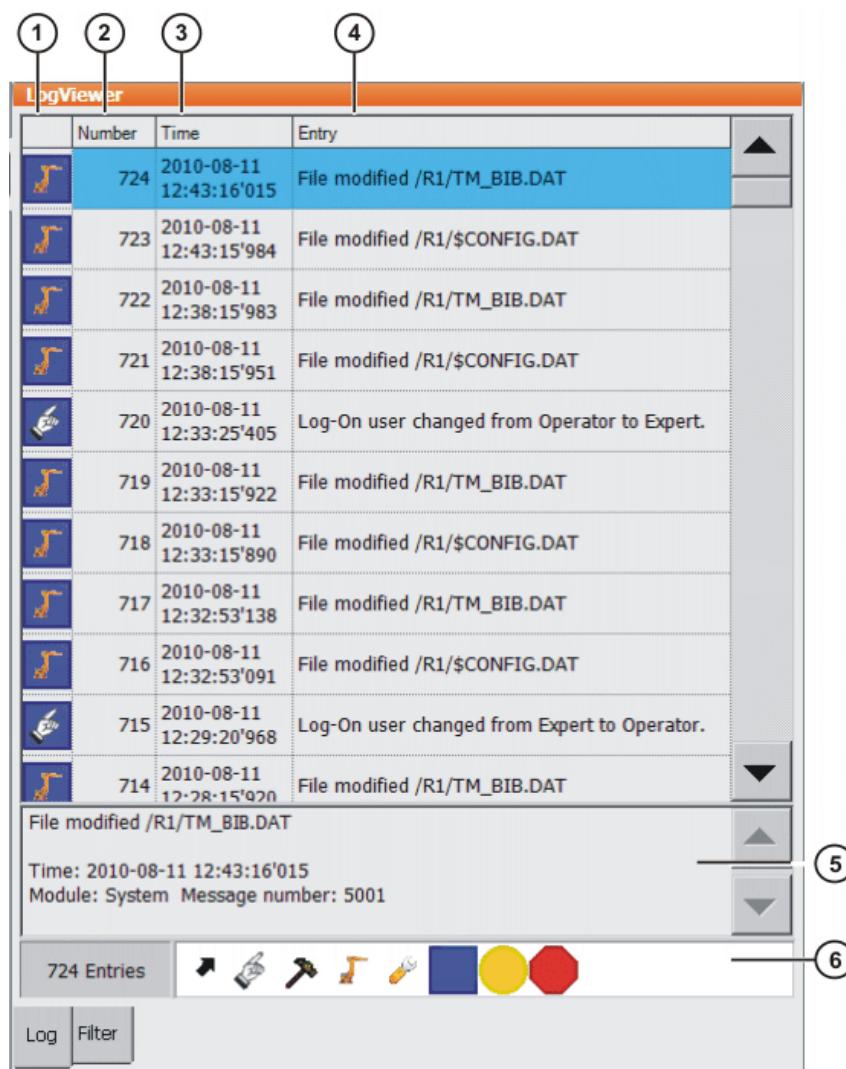


Fig. 13-1: Logbook, Log tab

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 <b>Filter</b> 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
<b>Export</b>	Exports the log data as a text file.  (>>> 13.1.4 "Configuring the logbook" Page 483)
<b>Update</b>	Refreshes the log display.

### 13.1.3 Filter tab

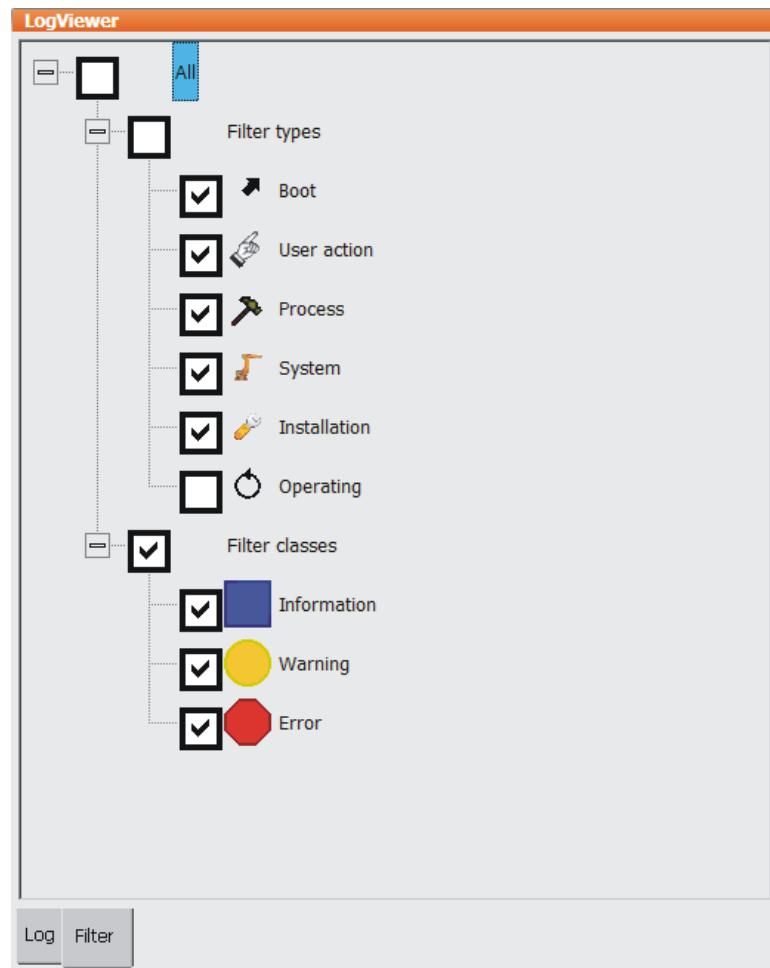


Fig. 13-2: Logbook, Filter tab

### 13.1.4 Configuring the logbook

- Precondition**
- "Expert" user group
- 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**

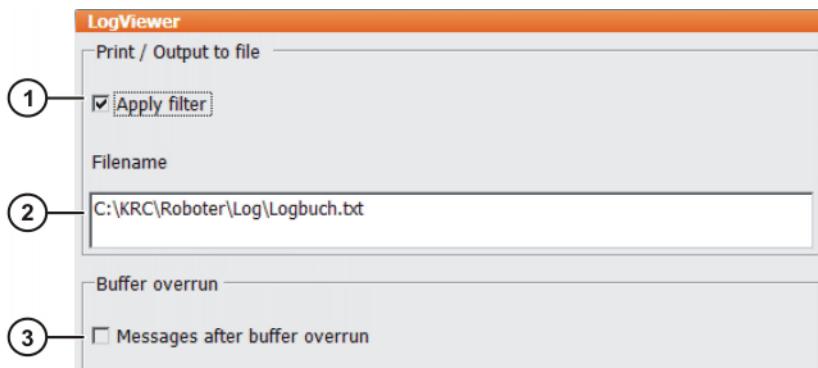


Fig. 13-3: Logbook configuration window

Item	Description
1	<ul style="list-style-type: none"> <li>■ <b>Check box active:</b> the log events selected with the filter are saved in the text file.</li> <li>■ <b>Check box not active:</b> all log events are saved in the text file.</li> </ul>
2	<p>Enter the path and name of the text file. Default path: C:\KRC\ROBOTER\LOG\LOGBUCH.TXT</p>
3	<ul style="list-style-type: none"> <li>■ <b>Check box active:</b> log data deleted because of a buffer overflow are indicated in gray in the text file.</li> <li>■ <b>Check box not active:</b> log data deleted because of a buffer overflow are not indicated in the text file.</li> </ul>

### 13.2 Displaying the caller stack

This function displays the data for the process pointer (\$PRO\_IP).

- Precondition**
- User group "Expert"
  - Program is selected.
- Procedure**
- In the main menu, select **Diagnosis > Caller stack**.

## Description

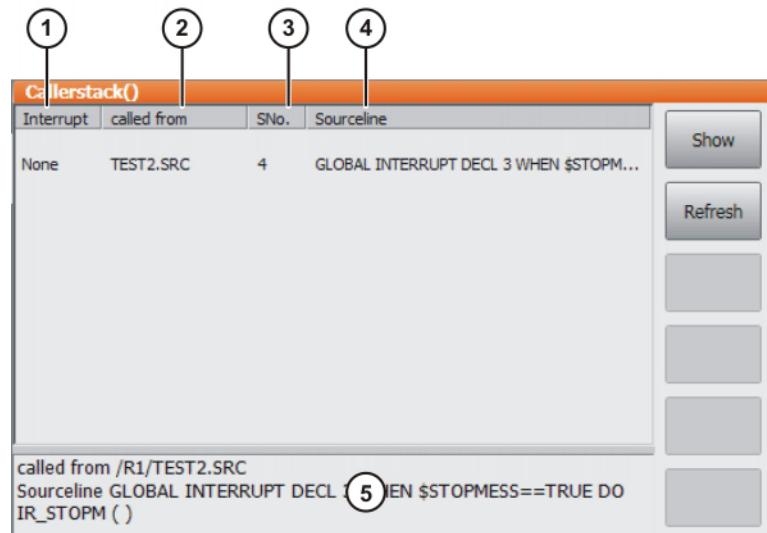
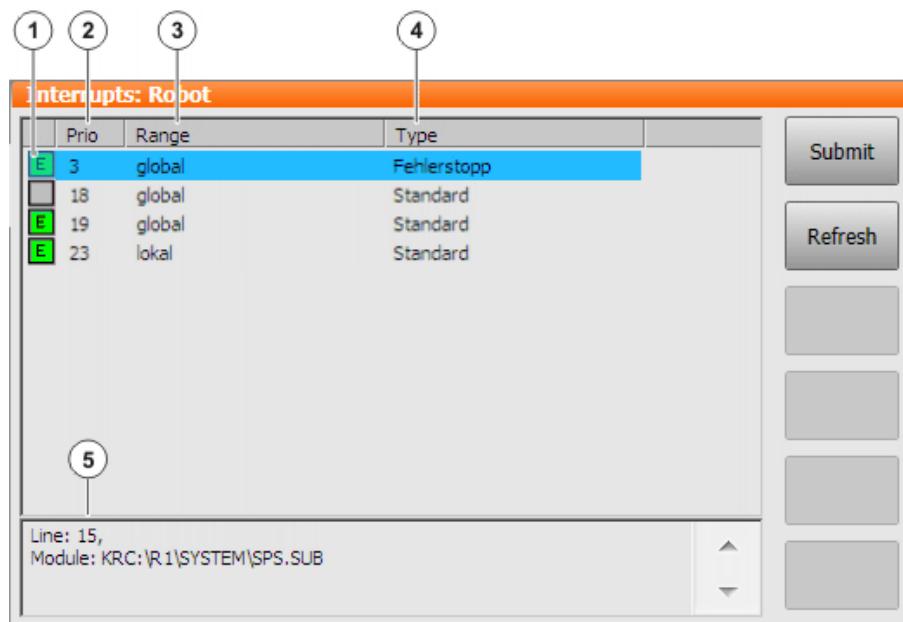


Fig. 13-4: Caller Stack window

Item	Description
1	<ul style="list-style-type: none"> <li>■ <b>None:</b> Call not initiated by interrupt</li> <li>■ <b>[No.]:</b> Call initiated by interrupt with the number [No.]</li> </ul>
2	This file contains the call.
3	<p>The program line with this number contains the call. Preconditions in the program for the correct line to be determined using the number:</p> <ul style="list-style-type: none"> <li>■ Detail view (ASCII mode) is activated.</li> <li>■ All Point PLCs are open.</li> </ul>
4	Source line
5	Detailed information about the entry selected in the list

### 13.3 Displaying interrupts

- Precondition**
- “Expert” user group
- Procedure**
- In the main menu, select **Diagnosis > Interrupts**.

**Description****Fig. 13-5: Interrupts**

Item	Description
1	Status of the interrupt <ul style="list-style-type: none"> <li>■ <span style="background-color: green; border: 1px solid black; padding: 2px;">E</span> Interrupt ON or ENABLE</li> <li>■ <span style="background-color: red; border: 1px solid black; padding: 2px;">D</span> Interrupt DISABLE</li> <li>■ <span style="background-color: lightgray; border: 1px solid black; padding: 2px;"></span> Interrupt OFF or not activated</li> </ul>
2	Number/priority of the interrupt
3	Validity range of the interrupt: global or local
4	Type of interrupt, dependent on the defined event in the interrupt declaration <ul style="list-style-type: none"> <li>■ <b>Standard:</b> e.g. \$IN[...]</li> <li>■ <b>Error stop:</b> \$STOPMESS</li> <li>■ <b>EMERGENCY STOP:</b> \$ALARM_STOP</li> <li>■ <b>Measurement (Fast Measurement):</b> \$MEAS_PULSE[1...5]</li> <li>■ <b>Trigger:</b> Trigger subprogram</li> </ul>
5	Module and program line of the interrupt declaration

The following buttons are available:

Button	Description
<b>Submit/ Robot</b>	Toggles between the displays for robot interrupts and Submit interrupts.
<b>Refresh</b>	Refreshes the display.

**13.4 Displaying diagnostic data about the kernel system****Description**

The menu item **Diagnostic monitor** makes it possible to display a wide range of diagnostic data concerning numerous software sub-areas of the kernel system.

Examples:

- Area **Kcp3 driver** (= driver for the smartPAD)

- Network driver

The data displayed depend on the selected area. The display includes states, fault counters, message counters, etc.

**Procedure**

1. In the main menu, select **Diagnosis > Diagnostic monitor**.
2. Select an area in the **Module** box.

Diagnostic data are displayed for the selected area.

### 13.5 Automatically compressing data for error analysis (KRCDiag)

**Description**

If it is necessary for an error to be analyzed by KUKA Deutschland GmbH, 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 Deutschland GmbH 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"**

- 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 > [...]**. In this way, the data can be stored on a USB stick or network path.

(>>> 7.10 "Archiving and restoring data" Page 253)

## 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. ([>>> 15 "KUKA Service" Page 495](#))

### 14.1 System requirements

The System Software 8.3 can be run on the following robot controller:

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

### 14.2 Installing Windows and the KUKA System Software (KSS) (from image)

#### Description

There are several variants for loading and finalizing the image. The most commonly required procedure is described here. The procedure also illustrates the step in which a master image can be created if required.



Information about the other variants and related issues is contained in the following documentation:

- **KUKA.RecoveryUSB** documentation: information about the creation and restoration of images, configuration of the stick, and the possible modes
- **WES7 System Preparation** expert documentation: information about finalizing and creating master images

#### Precondition

- Bootable KUKA USB stick with **KUKA.RecoveryUSB** software and an image
- The stick has been configured with “Silent” mode active.
- 2 GB RAM
- The robot controller is switched off.



We strongly recommend installing the desired software options (e.g. technology packages) before **Finalize Installation** is started by selecting **Execute**. The procedure below follows this approach.

During the finalizing process, the projects of the robot controller are rebuilt on the basis of the active project. Only the options already installed by this point are later included in all projects (active project, initial project and base project).



The LEDs on the CSP provide information about the installation status. Information about the LEDs is contained in the **KUKA.RecoveryUSB** documentation.

#### Procedure

1. Connect the USB stick to the robot controller.
2. Switch on the robot controller. The installation starts automatically.

Observe the LEDs on the CSP! Initially, there is no image displayed on the smartPAD.

3. Once the LEDs indicate that the Windows installation has been completed, remove the stick.
  - Following installation of Windows, the robot controller automatically reboots.
  - After this, the robot controller automatically reboots again a second time. (By now, at the latest, the stick must have been removed.)
4. The dialog **Finalize Installation** is displayed.

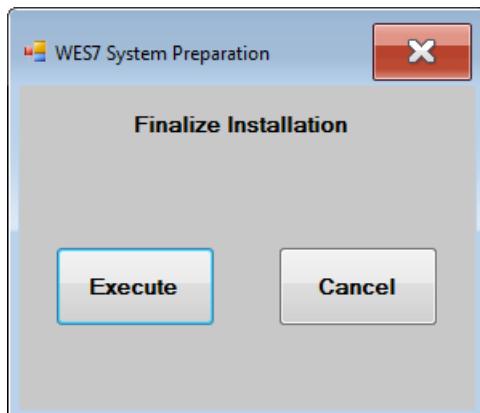


Fig. 14-1: Finalize Installation dialog box

5. Click on **Cancel** in the **Finalize Installation** dialog box.

A message informs the user that the dialog box will appear again the next time the system is started.

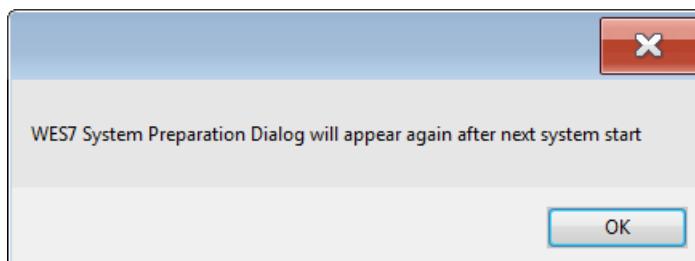


Fig. 14-2: Message after Cancel

6. Confirm the message with **OK**.

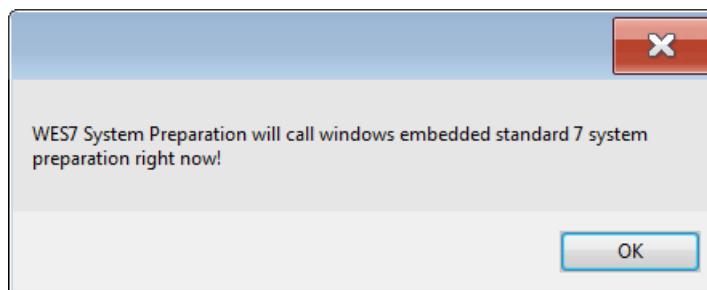
7. Install the desired software options.

If required, the robot controller can be rebooted between the individual installations. Following each start, the **Finalize Installation** dialog box must then again be answered with **Cancel**.

8. Once the last software option has been installed, reboot the robot controller.

9. Now select **Execute** in response to the **Finalize Installation** dialog box.

A message informs the user that the system preparation will now be started.



**Fig. 14-3: Message after Execute**

10. Confirm the message with **OK**.



If a master image is to be created: Do not yet plug in a USB stick. Plug in the stick only after the second shutdown – as described below.

The Generalize Phase is executed. After this, the robot controller reboots automatically and then shuts down automatically.

After the shutdown:

- If a master image is to be created: Continue with step 11.
- If not: Continue with step 12.

11. Only execute this step (step 11) if a master image is to be created.

- a. Plug in the USB Recovery Stick.



The configuration of the stick must now be such that an image can be created!

- b. Reboot the robot controller.
- c. Create the image.

In GUI mode, the image must be created via the user interface. In Silent mode, image creation starts automatically.

Once the image has been created, the robot controller shuts down automatically.

- d. Once the robot controller has shut down: Remove the USB stick.

12. Now reboot the robot controller.

The Specialize Phase is executed. This causes the robot controller to automatically reboot twice.

13. Now Mini-Setup starts:

Select the desired language. Confirm with **Next**.

14. Information about the installation and copyright is displayed. Confirm with **Next**.

15. Specify whether the robot controller is an **OPS** (Offline Programming System), also called “Office PC”. This is generally not the case, i.e. do not activate check box. Confirm with **Next**.

16. The system suggests a robot type. Confirm with **Next**.

Or: If the suggested type does not correspond to the type that is being used, select a different type. Then confirm with **Next**.

17. A summary of the setup settings is displayed. Confirm with **Next**.

The **Initial Project Setup** phase is executed. (This phase is short.)

The robot controller then shuts down automatically.

The robot controller can now be restarted in order to load the active project in WorkVisual and configure it.



The project contains the KR 210 robot by default. Make sure to remove the robot from the tree structure in WorkVisual and insert the robot actually being used.

This exchange must also be performed if the robot actually being used is already the KR 210.

Once the project has been configured in WorkVisual, it can again be transferred to the robot controller.

If necessary, the computer name can now be changed.

(>>> 14.3 "Changing the computer name" Page 490)

### 14.3 Changing the computer name

- |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b>  | KUKA.RecoveryUSB assigns a new computer name whenever an image is restored. The name can be changed at any point after the finalizing process.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Precondition</b> | <ul style="list-style-type: none"><li>■ User group “Expert”</li><li>■ Operating mode T1 or T2.</li><li>■ No program is selected.</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>Procedure</b>    | <ol style="list-style-type: none"><li>1. In the main menu, select <b>Start-up &gt; Additional software</b>.<br/>The <b>Installed additional software</b> window is opened.</li><li>2. Press the <b>New software</b> button.<br/>The <b>Selection</b> window is opened.</li><li>3. Press the <b>Configure</b> button.<br/>The <b>Configure installation paths</b> window is opened.</li><li>4. Select a line in the <b>Installation paths for options</b> area.<br/><b>Note:</b> If the line already contains a path, this path will be overwritten.</li><li>5. Press <b>Path selection</b>. The available drives are displayed.</li><li>6. Navigate to C:\KUKA and mark the <b>WES7RenameComputer</b> folder there.</li><li>7. Press <b>Save</b>. The <b>Configure installation paths</b> window is again displayed. It now contains the new path.</li><li>8. Mark the line with the new path and again press <b>Save</b>.<br/>The <b>Selection</b> window is again displayed.</li><li>9. Mark the <b>WES7RenameComputer</b> entry.</li><li>10. Press <b>Install</b>. Answer the request for confirmation with <b>Yes</b>.</li><li>11. The <b>W7RenCom</b> window and a popup keyboard are displayed.</li><li>12. Enter the desired name and press the <b>Set Computername</b> button to confirm.</li><li>13. The following message is displayed: <i>To finish the name setting you need to reboot the PC</i>.<br/>Confirm the message with <b>OK</b>.</li><li>14. Reboot the robot controller.</li></ol> |

### 14.4 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.

- |                     |                                                                                                                                 |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------|
| <b>Precondition</b> | <ul style="list-style-type: none"><li>■ “Expert” user group</li><li>■ T1 or T2 mode</li><li>■ No program is selected.</li></ul> |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------|

- 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.

**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:

<b>Button</b>	<b>Description</b>
<b>New software</b>	All programs available for installation are displayed.
<b>Back</b>	Additional software already installed is displayed.
<b>Refresh</b>	Refreshes the display, e.g. after a USB stick has been connected.
<b>Install</b>	Displays additional buttons: <ul style="list-style-type: none"> <li>■ <b>Yes</b>: The selected software is installed. If it is necessary to reboot the controller, this is indicated by a message.</li> <li>■ <b>No</b>: The software is not installed.</li> </ul>

Button	Description
<b>Configure</b>	<p>This button is only displayed if <b>New software</b> has been pressed.</p> <p>Paths for the installation of additional software or for updates of the System Software can be selected and saved here.</p> <p>Displays additional buttons:</p> <ul style="list-style-type: none"> <li>■ <b>Path selection:</b> A new path can be selected.</li> <li>■ <b>Save:</b> Saves the displayed paths.</li> </ul>
<b>Uninstall</b>	<p>Displays additional buttons:</p> <ul style="list-style-type: none"> <li>■ <b>Yes:</b> The selected software is uninstalled.</li> <li>■ <b>No:</b> The software is not uninstalled.</li> </ul>

## 14.5 KSS update

<b>Description</b>	<p>This function can be used to install KSS updates, e.g. from KSS 8.3.0 to KSS 8.3.1.</p> <p>Following installation or update of the KUKA System Software, the robot controller always performs an initial cold start.</p> <p>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.</p>
--------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------



Do not use this function to install a new version, e.g. from KSS 8.2 to KSS 8.3. Nor may this function be used to install a variant, e.g. from KSS 8.3 to KSS 8.3 sr. 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.

<b>Overview</b>	<p>There are 2 ways of installing a KSS update:</p> <ul style="list-style-type: none"> <li>■ From USB memory stick (&gt;&gt;&gt; 14.5.1 "Update from USB stick" Page 492)</li> <li>■ From the network (&gt;&gt;&gt; 14.5.2 "Update from the network" Page 493)</li> </ul>
-----------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

### 14.5.1 Update from USB stick

<b>NOTICE</b>	<p>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.</p>
---------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------

<b>Precondition</b>	<ul style="list-style-type: none"> <li>■ "Expert" user group</li> <li>■ T1 or T2 mode</li> <li>■ No program is selected.</li> <li>■ USB stick with the software to be installed <ul style="list-style-type: none"> <li>■ ZIP files must be unzipped.</li> <li>■ There must be no other files in the directory in which the individual files are located.</li> </ul> </li> </ul>
<b>Procedure</b>	<ol style="list-style-type: none"> <li>1. Plug in USB stick.</li> </ol>

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 following message is now displayed in the message window: *To complete software update, please REBOOT computer!*
4. Select **Shutdown** in the main menu and then the option **Reboot control PC (Reload files is not necessary)**.
5. Confirm the request for confirmation with **Yes**. The robot controller is rebooted and performs the update.  
The robot controller then reboots again.
6. Once the robot controller has rebooted for the second time, the USB stick can be removed.

The updated System Software is now available.

#### 14.5.2 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** For the preparation:

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

For the procedure:

- No program is selected.
- T1 or T2 operating mode

**Preparation** Configure the network path from which the update installation is to be carried out:

1. In the main menu, select **Start-up > Additional software**.
2. Press **New software**.
3. Press **Configure**.
4. Select the **Installation path for KRC update via the network** box. Press **Path selection**.
5. Select the desired network path (= the directory in which the Setup.exe file is located). Press **Save**.
6. The selected path is now displayed in the **Installation path for KRC update via the network** box.  
Press **Save** again.
7. Close the window.



It is only necessary to configure the network path once. It remains saved for subsequent updates.

**Procedure**

1. In the main menu, select **Start-up > Software update > Net**.
2. A request for confirmation is displayed, asking if the update should be carried out. Confirm by pressing **Yes**.

Depending on the network utilization, the procedure may take up to 15 min.

3. A message is displayed, indicating that a cold start will be forced next time the system is booted. Switch the controller off.
4. Wait until the computer has shut down completely. Then switch the controller back on.
5. Once the update has been completed, the computer is automatically shut down and rebooted.

## 15 KUKA Service

### 15.1 Requesting support

**Introduction** This documentation provides information on operation and operator control, and provides assistance with troubleshooting. For further assistance, please contact your local KUKA subsidiary.

**Information** **The following information is required for processing a support request:**

- Description of the problem, including information about the duration and frequency of the fault
- As comprehensive information as possible about the hardware and software components of the overall system

The following list gives an indication of the information which is relevant in many cases:

- Model and serial number of the kinematic system, e.g. the manipulator
- Model and serial number of the controller
- Model and serial number of the energy supply system
- Designation and version of the system software
- Designations and versions of other software components or modifications
- Diagnostic package KRCDiag
  - Additionally for KUKA Sunrise: Existing projects including applications
  - For versions of KUKA System Software older than V8: Archive of the software (KRCDiag is not yet available here.)
- Application used
- External axes used

### 15.2 KUKA Customer Support

**Availability** KUKA Customer Support is available in many countries. Please do not hesitate to contact us if you have any questions.

**Argentina** Ruben Costantini S.A. (Agency)  
Luis Angel Huergo 13 20  
Parque Industrial  
2400 San Francisco (CBA)  
Argentina  
Tel. +54 3564 421033  
Fax +54 3564 428877  
[ventas@costantini-sa.com](mailto:ventas@costantini-sa.com)

**Australia** KUKA Robotics Australia Pty Ltd  
45 Fennell Street  
Port Melbourne VIC 3207  
Australia  
Tel. +61 3 9939 9656  
[info@kuka-robotics.com.au](mailto:info@kuka-robotics.com.au)  
[www.kuka-robotics.com.au](http://www.kuka-robotics.com.au)

<b>Belgium</b>	KUKA Automatisering + Robots N.V. Centrum Zuid 1031 3530 Houthalen Belgium Tel. +32 11 516160 Fax +32 11 526794 <a href="mailto:info@kuka.be">info@kuka.be</a> <a href="http://www.kuka.be">www.kuka.be</a>
<b>Brazil</b>	KUKA Roboter do Brasil Ltda. Travessa Claudio Armando, nº 171 Bloco 5 - Galpões 51/52 Bairro Assunção CEP 09861-7630 São Bernardo do Campo - SP Brazil Tel. +55 11 4942-8299 Fax +55 11 2201-7883 <a href="mailto:info@kuka-roboter.com.br">info@kuka-roboter.com.br</a> <a href="http://www.kuka-roboter.com.br">www.kuka-roboter.com.br</a>
<b>Chile</b>	Robotec S.A. (Agency) Santiago de Chile Chile Tel. +56 2 331-5951 Fax +56 2 331-5952 <a href="mailto:robotec@robotec.cl">robotec@robotec.cl</a> <a href="http://www.robotec.cl">www.robotec.cl</a>
<b>China</b>	KUKA Robotics China Co., Ltd. No. 889 Kungang Road Xiaokunshan Town Songjiang District 201614 Shanghai P. R. China Tel. +86 21 5707 2688 Fax +86 21 5707 2603 <a href="mailto:info@kuka-robotics.cn">info@kuka-robotics.cn</a> <a href="http://www.kuka-robotics.com">www.kuka-robotics.com</a>
<b>Germany</b>	KUKA Deutschland GmbH Zugspitzstr. 140 86165 Augsburg Germany Tel. +49 821 797-1926 Fax +49 821 797-41 1926 <a href="mailto:Hotline.robotics.de@kuka.com">Hotline.robotics.de@kuka.com</a> <a href="http://www.kuka.com">www.kuka.com</a>

<b>France</b>	KUKA Automatisme + Robotique SAS Techvallée 6, Avenue du Parc 91140 Villebon S/Yvette France Tel. +33 1 6931660-0 Fax +33 1 6931660-1 <a href="mailto:commercial@kuka.fr">commercial@kuka.fr</a> <a href="http://www.kuka.fr">www.kuka.fr</a>
<b>India</b>	KUKA Robotics India Pvt. Ltd. Office Number-7, German Centre, Level 12, Building No. - 9B DLF Cyber City Phase III 122 002 Gurgaon Haryana India Tel. +91 124 4635774 Fax +91 124 4635773 <a href="mailto:info@kuka.in">info@kuka.in</a> <a href="http://www.kuka.in">www.kuka.in</a>
<b>Italy</b>	KUKA Roboter Italia S.p.A. Via Pavia 9/a - int.6 10098 Rivoli (TO) Italy Tel. +39 011 959-5013 Fax +39 011 959-5141 <a href="mailto:kuka@kuka.it">kuka@kuka.it</a> <a href="http://www.kuka.it">www.kuka.it</a>
<b>Japan</b>	KUKA Japan K.K. YBP Technical Center 134 Godo-cho, Hodogaya-ku Yokohama, Kanagawa 240 0005 Japan Tel. +81 45 744 7531 Fax +81 45 744 7541 <a href="mailto:info@kuka.co.jp">info@kuka.co.jp</a>
<b>Canada</b>	KUKA Robotics Canada Ltd. 6710 Maritz Drive - Unit 4 Mississauga L5W 0A1 Ontario Canada Tel. +1 905 670-8600 Fax +1 905 670-8604 <a href="mailto:info@kukarobotics.com">info@kukarobotics.com</a> <a href="http://www.kuka-robotics.com/canada">www.kuka-robotics.com/canada</a>

<b>Korea</b>	KUKA Robotics Korea Co. Ltd. RIT Center 306, Gyeonggi Technopark 1271-11 Sa 3-dong, Sangnok-gu Ansan City, Gyeonggi Do 426-901 Korea Tel. +82 31 501-1451 Fax +82 31 501-1461 <a href="mailto:info@kukakorea.com">info@kukakorea.com</a>
<b>Malaysia</b>	KUKA Robot Automation (M) Sdn Bhd South East Asia Regional Office No. 7, Jalan TPP 6/6 Taman Perindustrian Puchong 47100 Puchong Selangor Malaysia Tel. +60 (03) 8063-1792 Fax +60 (03) 8060-7386 <a href="mailto:info@kuka.com.my">info@kuka.com.my</a>
<b>Mexico</b>	KUKA de México S. de R.L. de C.V. Progreso #8 Col. Centro Industrial Puente de Vigas Tlalnepantla de Baz 54020 Estado de México Mexico Tel. +52 55 5203-8407 Fax +52 55 5203-8148 <a href="mailto:info@kuka.com.mx">info@kuka.com.mx</a> <a href="http://www.kuka-robotics.com/mexico">www.kuka-robotics.com/mexico</a>
<b>Norway</b>	KUKA Sveiseanlegg + Roboter Sentrumsvegen 5 2867 Hov Norway Tel. +47 61 18 91 30 Fax +47 61 18 62 00 <a href="mailto:info@kuka.no">info@kuka.no</a>
<b>Austria</b>	KUKA Roboter CEE GmbH Gruberstraße 2-4 4020 Linz Austria Tel. +43 7 32 78 47 52 Fax +43 7 32 79 38 80 <a href="mailto:office@kuka-roboter.at">office@kuka-roboter.at</a> <a href="http://www.kuka.at">www.kuka.at</a>

<b>Poland</b>	KUKA Roboter CEE GmbH Poland Spółka z ograniczoną odpowiedzialnością Oddział w Polsce Ul. Porcelanowa 10 40-246 Katowice Poland Tel. +48 327 30 32 13 or -14 Fax +48 327 30 32 26 ServicePL@kuka-roboter.de
<b>Portugal</b>	KUKA Robots IBÉRICA, S.A. Rua do Alto da Guerra n° 50 Armazém 04 2910 011 Setúbal Portugal Tel. +351 265 729 780 Fax +351 265 729 782 info.portugal@kukapt.com www.kuka.com
<b>Russia</b>	KUKA Russia OOO 1-y Nagatinskiy pr-d, 2 117105 Moskau Russia Tel. +7 495 665-6241 support.robotics.ru@kuka.com
<b>Sweden</b>	KUKA Svetsanläggningar + Robotar AB A. Odhnars gata 15 421 30 Västra Frölunda Sweden Tel. +46 31 7266-200 Fax +46 31 7266-201 info@kuka.se
<b>Switzerland</b>	KUKA Roboter Schweiz AG Industriestr. 9 5432 Neuenhof Switzerland Tel. +41 44 74490-90 Fax +41 44 74490-91 info@kuka-roboter.ch www.kuka-roboter.ch
<b>Slovakia</b>	KUKA Roboter CEE GmbH organizačná zložka Bojnická 3 831 04 Bratislava Slovakia Tel. +420 226 212 273 support.robotics.cz@kuka.com

<b>Spain</b>	KUKA Iberia, S.A.U. Pol. Industrial Torrent de la Pastera Carrer del Bages s/n 08800 Vilanova i la Geltrú (Barcelona) Spain Tel. +34 93 8142-353 <a href="mailto:comercial@kukarob.es">comercial@kukarob.es</a>
<b>South Africa</b>	Jendamark Automation LTD (Agency) 76a York Road North End 6000 Port Elizabeth South Africa Tel. +27 41 391 4700 Fax +27 41 373 3869 <a href="http://www.jendamark.co.za">www.jendamark.co.za</a>
<b>Taiwan</b>	KUKA Automation Taiwan Co. Ltd. 1F, No. 298 Yangguang ST., Nei Hu Dist., Taipei City, Taiwan 114 Taiwan Tel. +886 2 8978 1188 Fax +886 2 8797 5118 <a href="mailto:info@kuka.com.tw">info@kuka.com.tw</a>
<b>Thailand</b>	KUKA (Thailand) Co. Ltd. No 22/11-12 H-Cape Biz Sector Onnut Sukhaphiban 2 road, Prawet Bangkok 10250 Thailand Tel. +66 (0) 90-940-8950 <a href="mailto:HelpdeskTH@kuka.com">HelpdeskTH@kuka.com</a>
<b>Czech Republic</b>	KUKA Roboter CEE GmbH organizační složka Pražská 239 25066 Zdiby Czech Republic Tel. +420 226 212 273 <a href="mailto:support.robotics.cz@kuka.com">support.robotics.cz@kuka.com</a>
<b>Hungary</b>	KUKA Robotics Hungaria Kft. Fö út 140 2335 Taksony Hungary Tel. +36 24 501609 Fax +36 24 477031 <a href="mailto:info@kuka-robotics.hu">info@kuka-robotics.hu</a>

**USA**

KUKA Robotics Corporation  
51870 Shelby Parkway  
Shelby Township  
48315-1787  
Michigan  
USA  
Tel. +1 866 873-5852  
Fax +1 866 329-5852  
[info@kukarobotics.com](mailto:info@kukarobotics.com)  
[www.kukarobotics.com](http://www.kukarobotics.com)

**UK**

KUKA Robotics UK Ltd  
Great Western Street  
Wednesbury West Midlands  
WS10 7LL  
UK  
Tel. +44 121 505 9970  
Fax +44 121 505 6589  
[service@kuka-robotics.co.uk](mailto:service@kuka-robotics.co.uk)  
[www.kuka-robotics.co.uk](http://www.kuka-robotics.co.uk)



# Index

## Symbols

\_TYP 371  
 \_TYPE 372  
 "Expert" 361  
 #BSTEP 269  
 #CSTEP 270  
 #GO 269  
 #IGNORE 300, 301  
 #ISTEP 269  
 #MSTEP 269  
 #PSTEP 270  
 \$ 365  
 \$ACCU\_STATE 93  
 \$ADAP\_ACC 187, 193  
 \$ADVANCE 270  
 \$ALARM\_STOP 202, 203  
 \$ALARM\_STOP\_INTERN 202, 203  
 \$ANIN 349  
 \$ANOUT 349  
 \$AUT 204  
 \$BRAKE\_SIG 220  
 \$BRAKES\_OK 231  
 \$BRAKETEST\_MONTIME 230  
 \$BRAKETEST\_REQ\_EX 230  
 \$BRAKETEST\_REQ\_INT 230  
 \$BRAKETEST\_WARN 231  
 \$BRAKETEST\_WORK 231  
 \$BRK\_DEL\_COM 463  
 \$BRK\_DEL\_EX 463  
 \$BRK\_DEL\_PRO 463  
 \$BRK\_MODE 463  
 \$BWD\_INFO 281  
 \$BWDSTART 281  
 \$CHCK\_MOVENA 200  
 \$CIRC\_MODE 302  
 \$CIRC\_TYPE 302  
 \$COLL\_ALARM 188  
 \$COLL\_ENABLE 188  
 \$CONF\_MESS 201  
 \$CONST\_VEL 396  
 \$CONST\_VEL\_C 397  
 \$COOLDOWN\_TIME 186  
 \$COULD\_START\_MOTION 52  
 \$DIST\_LAST 446  
 \$DIST\_NEXT 446  
 \$DRIVES\_OFF 201  
 \$DRIVES\_ON 201  
 \$ECO\_LEVEL 176  
 \$ERR 404  
 \$EX\_AX\_IGNORE 398  
 \$EXT 204  
 \$EXT\_START 200  
 \$HOLDING\_TORQUE 221  
 \$I\_O\_ACT 201  
 \$I\_O\_ACTCONF 202  
 \$IN 349  
 \$IN\_HOME 203  
 \$LDC\_CONFIG 153

\$LDC\_LOADED 151  
 \$LDC\_RESULT 153  
 \$LOAD\_BWINI 281  
 \$MOVE\_ENABLE 200  
 \$NEAR\_POSRET 203  
 \$ON\_PATH 203  
 \$ORI\_TYPE 286, 300  
 \$OUT 349  
 \$PAL\_MODE 102  
 \$PATHTIME 394  
 \$PERI\_RDY 52, 202  
 \$POS\_ACT 448  
 \$PRO\_ACT 203  
 \$PRO\_IP 483  
 \$PRO\_MODE 269  
 \$PRO\_MOVE 203  
 \$SRC\_RDY1 202  
 \$ROB\_CAL 202  
 \$ROB\_STOPPED 203  
 \$ROBRUNTIME 90, 91  
 \$SPL\_ORI\_JOINT\_AUTO 301  
 \$STOP\_CONST\_VEL\_RED 396  
 \$STOPMESS 202  
 \$T1 204  
 \$T2 204  
 \$TOOL\_DIRECTION 126  
 \$TORQ\_DIFF 188, 192, 193  
 \$TORQ\_DIFF2 188  
 \$TORQMON\_COM\_DEF 188  
 \$TORQMON\_DEF 188  
 \$TORQMON\_TIME 188, 193  
 \$TORQUE\_AXIS\_ACT 219, 221  
 \$TORQUE\_AXIS\_LIMITS 220  
 \$TORQUE\_AXIS\_MAX 220  
 \$TORQUE\_AXIS\_MAX\_0 220  
 \$US2\_VOLTAGE\_ON 99  
 \$USER\_SAF 53, 202  
 \$VW\_BACKWARD 281  
 \$VW\_CYCFLAG 281  
 \$VW\_MOVEMENT 281  
 \$VW\_RETRACE\_AMF 281  
 \$WARMUP\_CURR\_LIMIT 186  
 \$WARMUP\_MIN\_FAC 186  
 \$WARMUP\_RED\_FAC\_ACT 186  
 \$WARMUP\_RED\_VEL 185  
 \$WARMUP\_SLEW\_RATE 186  
 \$WARMUP\_TIME 185

## Numbers

2006/42/EU2006 42  
 2014/30/EU2014 42  
 2014/68/EU2014 42  
 3-point method 133  
 95/16/EC 42

## A

A6, mastering position 117  
 ABC 2-point method 131

ABC World method 130  
Absolute value (mathematical function) 455  
Accessories 17, 19  
Activation, project 257  
Actual position 79  
Addition 447  
Addition, geometric 448  
Administrator 63  
Advance run 270  
Advance run stop 285, 399  
Analog inputs 412  
Analog outputs 413  
ANIN 412  
ANOUT 351, 413  
ANSI/RIA R.15.06-2012 43  
APPL\_RUN 203  
Applied norms and regulations 42  
Approximate positioning 285, 318  
Approximate positioning, homogenous 442  
Approximate positioning, mixed 443  
Arc cosine (mathematical function) 455  
Arc sine 456  
Arc tangent (mathematical function) 455  
Archiving overview 253  
Archiving, logbook 255  
Archiving, network 255  
Archiving, to USB stick 254  
Areas of validity 367  
Automatic mode 39  
Auxiliary point 284, 374, 377  
Axis limitation, mechanical 30  
Axis monitoring functions, configuring 165  
Axis range 20

**B**

Backup configuration (window) 264  
Backup Manager 260  
Backup Manager, configuring 264  
Backup, option packages 260, 261  
Backup, projects 260, 261  
Backup, RDC data 260, 261  
Backward motion (using "Start backwards" key) 276  
Backward motion, configuring 194  
Backward motion, prevention 457  
BACKWARD\_STEP 195  
Base calibration 133  
BASE coordinate system 64, 133  
Battery state 93  
Bit operators 452  
Block pointer 244, 270  
Block selection 274, 292  
BRAKE 426, 432, 433  
Brake defect 32  
Brake delay 462  
BRAKE F 433  
BRAKE FF 433  
Brake release device 31  
Brake test 227  
Brake test cycle time 228  
Brake test, function test 234

Brake test, programs 229  
Brake test, signals 230, 231  
Brake test, teaching positions 233  
Brake, defective 233  
BrakeTestBack.SRC 229, 233  
BrakeTestPark.SRC 229, 233  
BrakeTestReq.SRC 229, 234  
BrakeTestSelfTest.SRC 229, 235  
BrakeTestStart.SRC 229, 233  
Braking distance 20  
Branch, conditional 402  
BSTEP 269  
Bypassing workspace monitoring 76

**C**

Calibrating an external kinematic system 144  
Calibration 126  
Calibration points (menu item) 89  
Calibration tolerances, defining 193  
Calibration, base 133  
Calibration, external TCP 136  
Calibration, fixed tool 136  
Calibration, linear unit 142  
Calibration, root point, kinematic system 144  
Calibration, tool 126  
Calibration, TOOL kinematic system 148  
Calibration, workpiece 136  
Call by Reference 421  
Call by Value 421  
Caller stack 483  
Caller stack (menu item) 483  
Cancel, program 243  
CASE 409  
CAST\_FROM 447  
CAST\_TO 447  
CCLOSE 447  
CE mark 20  
CELL.SRC 275  
CHANNEL 447  
Checksum , safety configuration 172  
CIOCTL 447  
CIRC 374  
CIRC motion 315  
CIRC\_REL 377  
CIRC, motion type 284  
Circular angle 306  
Circular motion 374, 377  
Cleaning work 40  
Close all FOLDs (menu item) 249  
Cold start 57  
Cold start, initial 55, 57, 492  
Collision detection 186, 188, 317  
Collision detection (menu item) 188, 189  
Collision detection, Automatic External 190  
Collision detection, offset 189  
Collision detection, system variables 187  
Collision detection, variable 190  
Comment 250  
Comparison, data from kernel system and hard drive 227  
Conditional branch 402

Configuration 159  
 Configuration (menu item) 174  
 Configuring CELL.SRC 196  
 Connecting cables 17, 19  
 Connection manager 46  
 CONST 369  
 CONST\_VEL 395  
 Constant velocity range 330, 340, 395  
 Constants 368, 369  
 CONTINUE 399  
 Continuous Path 283  
 Coordinate system for jog keys 50  
 Coordinate system for Space Mouse 49  
 Coordinate systems 64  
 Coordinate systems, angles 65  
 Coordinate systems, orientation 65  
 COOPEN 447  
 Copy 252  
 Cosine (mathematical function) 455  
 Counterbalancing system 40  
 Counters, displaying 87  
 CP motions 283  
 CP spline block 319, 380  
 CREAD 447  
 Creating a new folder 237  
 Creating a new program 237  
 Cut 252  
 CWRITE 447

**D**

Danger zone 20  
 DAT 364  
 Data list 364  
 Data type, user-defined 368, 370, 371  
 Data types 366  
 DECL 369  
 Declaration of conformity 20  
 Declaration of incorporation 19, 20  
 Decommissioning 41  
 DEF line (menu item) 247  
 DEF line, displaying/hiding 247  
 DEFAULT 409  
 DEFFCT ... ENDFCT 420  
 Delay time, power failure 58  
 Delay time, power-off 56, 58  
 DELETE\_BACKWARD\_BUFFER 457  
 Deleting mastering 123  
 Detail view (ASCII) (menu item) 247  
 Detail view, activating 247  
 Diagnosis 481  
 Diagnostic monitor (menu item) 486  
 Dial gauge 114  
 Directory structure 238  
 Display (menu item) 84  
 Displaying a variable, single 82, 83  
 Displaying the logbook 481  
 Displaying variables, in overview 84  
 Displaying, robot controller information 90  
 Displaying, robot information 90  
 Disposal 41  
 DISTANCE 435

Division 447  
 Documentation, industrial robot 15  
 Drive bus 57  
 Drives, switching on/off 53

**E**

EC declaration of conformity 20  
 Edit (button) 50  
 Editor 243  
 Electromagnetic compatibility (EMC) 43  
 ELSE 402  
 EMC Directive 20, 42  
 EMERGENCY STOP 46  
 EMERGENCY STOP device 27, 28, 32  
 EMERGENCY STOP, external 28, 35  
 EMERGENCY STOP, local 35  
 EN 60204-12006/A12009 43  
 EN 61000-6-22005 43  
 EN 61000-6-42007 + A12011 43  
 EN 614-12006+A12009 43  
 EN ISO 10218-12011 43  
 EN ISO 121002010 43  
 EN ISO 13849-12015 42  
 EN ISO 13849-22012 42  
 EN ISO 138502015 42  
 Enabling device 28, 32  
 Enabling device, external 29  
 Enabling switch 47  
 Enabling switches 28  
 ENDFCT 420  
 ENDFOR 400  
 ENDIF 402  
 Endless loop 403  
 ENDLOOP 403  
 ENDSPLINE 380, 382  
 ENDSWITCH 409  
 ENDWHILE 411  
 Energy consumption, measuring 77  
 ENUM 370  
 Enumeration type 370  
 ERR\_RAISE 403  
 EtherNet/IP interfaceWindows interface 161  
 Even parity 201  
 EXIT 400, 403  
 Exiting, KSS 55  
 Expert Submit (template) 476  
 Export (button) 172  
 External axes 19, 22, 79, 90  
 External kinematic system, calibration 144

**F**

FALSE 419  
 Faults 33  
 Field bus interface 161  
 Field bus, Ethernet-based 159  
 File list 238  
 File, properties 239  
 Filter 239  
 Find 252  
 First mastering 109, 118  
 Fixed tool, calibration 136

Flags, displaying 85, 86  
FLANGE coordinate system 65, 127  
Folder, creating 237  
Folder, properties 239  
Folds 248  
Folds, creating 251  
Folds, displaying 248  
Fonts 364  
FOR 410  
FOR ... TO ... ENDFOR 400  
FORWARD() 458  
Frame operation 448  
Function test 34  
Function, calling 419  
Function, syntax 420

**G**

General safety measures 32  
Geometric addition 448  
Global 367  
GLOBAL (interrupt declaration) 426  
GO (program run mode) 269  
GOTO 401

**H**

HALT 402  
Hardware, options 98  
Hazardous substances 40  
Header 238  
Help, messages 59  
Hibernate 57  
HOME position 246  
Homogenous approximate positioning 442  
HOV 70

**I**

I/O driver, reconfiguring 165  
I/Os, reconfiguring 165  
Identification plate 47  
IF ... THEN ... ENDIF 402  
Impact 188, 189  
IN parameters 421  
Increment 75  
Incremental jogging 75  
Indirect method 135  
Industrial robot 17, 19  
Info (menu item) 90  
Inline forms 313  
Inputs/outputs, analog 81, 349  
Inputs/outputs, Automatic External 81, 197  
Inputs/outputs, digital 79, 349  
Installation 487  
Intended use 18, 19  
INTERN.ZIP 254, 255  
Interpolation mode 317, 322  
INTERRUPT 425  
Interrupt 425  
Interrupt declaration 425  
INTERRUPT DISABLE 429  
INTERRUPT ENABLE 429  
INTERRUPT OFF 428

INTERRUPT ON 428  
Interrupt program 425  
Interrupts 484  
Introduction 15  
INV\_POS() 459  
INVERSE 459  
IP addresses 159  
ISTEP 269

**J**

Jerk 322, 323, 327, 328, 333, 335  
Jog keys 46, 66, 71  
Jog mode 29, 32  
Jog mode "Jog keys" 68  
Jog mode "Space Mouse" 68  
Jog mode, activating 70  
Jog override 70  
Jogging, axis-specific 65, 71  
Jogging, Cartesian 65, 71, 75  
Jogging, external axes 76  
Jogging, robot 65  
Jump 401

**K**

Keyboard 46  
Keyboard key 46  
Keypad 50  
Keywords 364  
Kinematics group 50, 68  
KLI, configuring 159  
KRCDiag 486  
KRL syntax 361  
KUKA Customer Support 90, 495  
KUKA Line Interface, configuring 159  
KUKA Service 495  
KUKA smartHMI 49  
KUKA smartPAD 21, 45  
KUKA.Load 150  
KUKA.LoadDataDetermination 150

**L**

Labeling 31  
Language 59  
Liability 19  
LIN 373  
LIN motion 314  
LIN\_REL 376  
LIN, motion type 284  
Line break (menu item) 248  
Line mark for mastering 118  
Linear motion 373, 376  
Linear unit 19, 141  
Load data 150  
Logbook 481  
Logbook, configuring 483  
Long texts, exporting 153  
Long texts, importing 153  
LOOP ... ENDLOOP 403  
Loss of mastering 109, 113, 117, 122  
Low Voltage Directive 20

**M**

Machine data 35, 90, 91  
 Machine data, configuration 97  
 Machinery Directive 20, 42  
 Main menu, calling 54  
 Maintenance 39, 155  
 Manipulator 17, 19, 21  
 Manual mode 38  
 Marked region 253  
 Mastering 103  
 Mastering after maintenance work 115  
 Mastering marks 105, 106  
 Mastering methods 104  
 Mastering position, A6 117  
 Mechanical end stops 30  
 MEMD 104, 116  
 Message help 59  
 Message window 49  
 Messages, displaying help 60  
 Micro Electronic Mastering Device 104, 116  
 Minimizing KUKA smartHMI 53  
 Mixed approximate positioning 443  
 Mode selector switch 46  
 Modifying a logic instruction 360  
 Modifying a variable 82  
 Modifying coordinates 342  
 Modifying motion parameters 342  
 Modifying variables 84  
 Module 364  
 Monitoring functions, checking 170  
 Monitoring, physical safeguards 26  
 Monitoring, velocity 29  
 Motion conditions (window) 52  
 Motion programming, basic principles 283  
 Motion types 283  
 Motor, exchange 116  
 MSTEP 269  
 Multiplication 447

**N**

Name, archive 91  
 Name, control PC 90  
 Name, robot 90  
 Names 364  
 Navigator 238  
 Non-rejecting loop 408  
 Numeric entry, base 136  
 Numeric entry, external TCP 138  
 Numeric entry, external tool 149  
 Numeric entry, linear unit 143  
 Numeric entry, root point, kinematic system 146  
 Numeric input, tool 132

**O**

Odd parity 201  
 Offset 109, 112, 117, 121, 351  
 OLDC 151  
 ON\_ERROR\_PROCEED 403  
 Online documentation 59  
 Online load data check 151  
 Online optimizing 227

Open all FOLDs (menu item) 249  
 Opening a program 243  
 Operating hours 91  
 Operating hours meter 91  
 Operating mode after reconfiguration 165  
 Operating mode after start 55  
 Operating mode, changing 63  
 Operation 45  
 Operator 62  
 Operator safety 24, 26, 32, 53  
 Operator safety acknowledgement 99  
 Operator, geometric 448  
 Operators 23  
 Operators for bit operations 452  
 Operators for comparison operations 451  
 Operators, arithmetic 447  
 Operators, logic 452  
 Operators, priority 454  
 Option packages, backup 260, 261  
 Option packages, restoring 260  
 Options 17, 19  
 Orientation behavior, SCIRC 302  
 Orientation control, LIN, CIRC 286  
 Orientation control, spline 300  
 OUT 349  
 OUT parameters 421  
 Output, analog 351  
 Output, digital 349  
 Overload 32  
 Override 70, 273  
 Override (menu item) 77  
 Overriding, power failure 56, 58  
 Overview of the industrial robot 17

**P**

Palletizing robot 128  
 Palletizing robots 102, 132  
 Panic position 28  
 Parameters, transferring 421  
 Parity 201  
 Parity bit 201  
 Parity, even 201  
 Parity, odd 201  
 Password, changing 176  
 Paste 252  
 PATH 438  
 Payload data 150  
 Performance Level 25  
 Performing a manual brake test 233  
 Peripheral contactor 37, 99, 100  
 Personnel 22  
 PGNO\_FBIT 199  
 PGNO\_FBIT\_REF 202  
 PGNO\_LENGTH 199  
 PGNO\_PARITY 200  
 PGNO\_REQ 203  
 PGNO\_TYPE 199  
 PGNO\_VALID 200  
 Pinning 256, 260  
 Plant integrator 22  
 PLC\_ROB\_STOP\_RELEASE() 462

- PLC\_ROB\_STOP() 462  
Point correction, defining limits 182  
Point-to-point 283  
Point-to-point motion 373, 375  
Positionally accurate robot, checking activation 101  
Positioner 19, 144  
POV 273  
Power failure delay time 58  
Power failure, overriding 56, 58  
Power-off delay time 56, 58  
Pre-mastering position 105, 107  
Pressure Equipment Directive 40, 42  
Preventive maintenance work 40  
Printing, program 253  
Priority 426, 436, 440  
Probe 104  
Product description 17  
PROFenergy 77  
PROFINET interface 161  
Program execution 269  
Program execution control 399  
Program lines, deleting 252  
Program override 273  
Program run mode, selecting 269  
Program run modes 269  
Program, canceling 243  
Program, closing 244  
Program, creating 237  
Program, editing 249  
Program, opening 243  
Program, printing 253  
Program, selecting 243  
Program, starting 273, 274  
Program, stopping 274, 275  
Programmer 63  
Programming, Expert 361  
Programming, inline forms 313  
Programming, KRL syntax 361  
Programming, User 313  
Project management (window) 258  
Project, activation 257  
Project, inactive 259  
Projects, backup 260, 261  
Projects, restoring 260  
Properties, file or folder 239  
Protective equipment 29  
PTP 373  
PTP motion 313  
PTP spline block 319, 382  
PTP\_REL 375  
PTP SPLINE ... ENDSPLINE 382  
PTP, motion type 283  
PUBLIC 368  
PULSE 350, 414  
Pulse 350, 414  
Pulse, path-related 359
- R**  
RDC data backup 92  
RDC data, restoring 263
- RDC, data backup 260, 261  
RDC, exchange 116  
Re-teaching 342  
Reaction distance 20  
Recommissioning 34, 95  
Reference mastering 115  
REFLECT\_PROG\_NR 199  
Rejecting loop 411  
Release device 30  
Renaming a file 237  
Renaming a folder 237  
Renaming the base 140  
Renaming the tool 140  
Repair 39  
REPEAT ... UNTIL 408  
Replace 253  
Resetting a program 275  
Restoration, option packages 260  
Restoration, projects 260  
Restoring data 256  
Restoring RDC data 263  
RESUME 433  
Reteaching, defining limits 182  
RETURN 420  
ROB\_STOP\_RELEASE() 461  
ROB\_STOP() 461  
Robot controller 17, 19  
Robot data (menu item) 90  
Robot name, changing 91  
Robot name, display 51  
ROBROOT coordinate system 64  
Runtime variable 367
- S**  
Safe operational stop 21, 29  
Safeguards, external 31  
Safety 19  
Safety configuration, Checksum 172  
Safety configuration, export 172  
Safety configuration, import 172  
Safety controller 25  
Safety functions 24, 32  
Safety functions, overview 24  
Safety instructions 15  
Safety of machinery 42, 43  
Safety options 21  
Safety STOP 0 21  
Safety STOP 1 21  
Safety STOP 2 21  
Safety STOP 0 21  
Safety STOP 1 21  
Safety STOP 2 21  
Safety stop, external 29  
Safety zone 21, 23  
Safety, general 19  
SCIRC 383  
SCIRC motion, programming 334  
SCIRC segment, programming 323  
SCIRC\_REL 387  
Screenshot, smartPAD 59  
SCTLCRC.XML 172

SEC 411  
 Selecting a program 243  
 Selecting the base 71  
 Selecting the operating mode 24, 25  
 Selecting the tool 71  
 Selecting, range in program 252  
 SEMD 104, 109  
 Serial number 91  
 Service life 21, 90  
 SET\_BRAKE\_DELAY() 462  
 SET\_TORQUE\_LIMITS 215, 218  
 Shutdown (menu item) 55  
 SIGNAL 418  
 Signal diagrams 206  
 Signals, brake test 230, 231  
 Simulation 39  
 Sine (mathematical function) 455  
 Single (menu item) 82, 83, 192  
 Single point of control 41  
 Singularities 311  
 Singularity, CP spline 300, 301  
 Singularity, LIN/CIRC 286  
 SLIN 383  
 SLIN motion, programming 332  
 SLIN segment, programming 323  
 SLIN\_REL 386  
 smartHMI 17, 49  
 smartPAD 21, 33, 45  
 Soft axes 212  
 Software 17, 19  
 Software limit switches 30, 32, 123  
 Software limit switches, modifying 123  
 Space Mouse 46, 66, 72, 74, 75  
 Special characters 313  
 SPL 384  
 SPL segment, programming 323  
 SPL\_REL 388  
 SPLINE ... ENDSPLINE 380  
 Spline block, programming 319  
 Spline segment 289  
 Spline, motion type 289  
 SPOC 41  
 SPS.SUB, editing 475  
 SPTP 385  
 SPTP motion, programming 336  
 SPTP segment, programming 325  
 SPTP\_REL 389  
 Square root (mathematical function) 455  
 SRC 364  
 SREAD 447  
 Stamp 250  
 Standard Electronic Mastering Device 104, 109  
 Standstill monitoring 166  
 Start backwards key 46  
 Start key 46, 47  
 Start type, KSS 55  
 Start types 57  
 Start-up 34, 95  
 Start-up mode 37  
 Start-up wizard 97  
 Starting a program, automatic 274  
 Starting a program, manual 273  
 Starting Automatic External mode 275  
 Starting the KSS 54  
 Status 307  
 Status bar 49, 51, 238  
 Status keys 46  
 STEP 400  
 STOP 0 20, 21  
 STOP 1 20, 22  
 STOP 2 20, 22  
 Stop category 0 21  
 Stop category 1 22  
 Stop category 2 22  
 Stop category 1, Drive Ramp Stop 22  
 STOP key 46  
 Stop reactions 24  
 STOP WHEN PATH 397  
 STOP 1 - DRS 22  
 Stopping a program 274, 275  
 Stopping distance 20, 23  
 Stopping the robot 461  
 Stopping, robot 432  
 Storage 41  
 Storage capacities 90  
 String variable length after initialization 469  
 String variable length in the declaration 468  
 String variable, deleting contents 469  
 String variables 467  
 String variables, comparing contents 471  
 String variables, copying 472  
 String variables, extending 470  
 String variables, searching 470  
 STRUC 371  
 Structure type 371  
 SUB program, creating 476  
 Submit (template) 476  
 Submit interpreter 51, 473  
 Submit interpreter, editing SPS.SUB 475  
 Submit interpreter, starting 474  
 Submit interpreter, stopping 474  
 Subprogram, calling 419  
 Subtraction 447  
 Supplementary load data (menu item) 151  
 Support request 495  
 SWITCH ... CASE ... END SWITCH 409  
 Switching action, path-related 354  
 Switching on the robot controller 54  
 SWRITE 447  
 Symbols 364  
 SYN OUT 354  
 SYN PULSE 359  
 System integrator 20, 22, 23  
 System requirements 18, 487  
 System variables 280

**T**

T1 (operating mode) 22  
 T2 (operating mode) 22  
 Tangent (mathematical function) 455  
 TCP 126  
 TCP, external 136

Teach pendant 17, 19  
Teaching 342  
Technology packages 17, 90, 313, 365  
Terms used, safety 20  
Time block 392  
TIME\_BLOCK 392  
Timers, displaying 88  
tm\_useraction 187  
tm\_useraction, editing 191  
TMx 190  
Tool calibration 126  
Tool Center Point 126  
TOOL coordinate system 64, 126  
Tool direction 126  
Tool load data (menu item) 150  
Tool, external 148  
Torque 188, 189  
Torque mode, diagnosis 219  
Torque mode, examples 214, 222  
Torque mode, overview 212  
Torque monitoring 192  
Torque monitoring (menu item) 193  
Touch screen 45, 50  
Trademarks 16  
Training 15  
Transforming coordinates 343  
Transportation 33  
TRIGGER 435, 438  
Trigger, for spline inline form 328  
Turn 307, 310  
Turn-tilt table 19, 144  
Type, robot 90  
Type, robot controller 90

**U**

Unmastering 123  
UNTIL 408  
Update 492  
US2 37, 99, 100  
USB connection 47  
USB sticks 18  
Use, contrary to intended use 19  
Use, improper 19  
User 20, 22  
User group, changing 62  
User group, default 62  
User interface 49

**V**

Variable correction 82  
Variable overview, configuring 174  
VARSTATE() 83, 465  
Velocity 71, 273  
Velocity monitoring 29  
Version, kernel system 90  
Version, operating system 90  
Version, robot controller 90  
Version, user interface 90  
Voltage 81, 351, 352

**W**

WAIT 352, 410, 411  
WAIT FOR 410  
Wait function, signal-dependent 352  
WAIT SEC 411  
Wait time 352, 411  
WAITFOR 352  
Warm-up 183  
Warnings 15  
WHILE ... ENDWHILE 411  
Windows interface 53, 159  
WITH (permissible system variables 390  
Workpiece base, calibrating 146  
Workpiece base, numeric entry 148  
Workspace 20, 23  
Workspaces, axis-specific 177  
Workspaces, bypassing monitoring 76  
Workspaces, Cartesian 177  
Workspaces, mode 182  
WORLD coordinate system 64  
Wrist root point 182

**X**

XML export 172  
XML import 172  
XYZ 4-point method 128  
XYZ Reference method 129

